Chorus:une architecture pour les systemes repartis

M. Guillemont, H. Zimmermann, G. Morisset, J.S. Banino

INRIA

CENTRE DE ROCQUENCOURT

Rapports de Recherche

N° 274

CHORUS:
UNE ARCHITECTURE POUR LES SYSTÈMES RÉPARTIS

Marc GUILLEMONT
Hubert ZIMMERMANN
Gérard MORISSET
Jean-Serge BANINO

Mars 1984

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tél. (3) 954 90 20

CHORUS:

Une architecture pour les systèmes répartis

M. GUILLEMONT **

H. ZIMMERMANN *

G. MORISSET **

J.S. BANINO **

* CNET
38, 40 rue du Général Leclerc
92121 Issy les Moulineaux

** INRIA
BP 105
78153 Le Chesnay Cedex

RESUME

CHORUS est une architecture de système réparti conçue pour une grande variété de machines et d'applications. Bâtie autour d'un petit ensemble de concepts simples et puissants, CHORUS est une base solide et saine pour le développement d'applications réparties, l'apprentissage de la répartition et la recherche.

CHORUS propose une approche originale de la répartition où les communications jouent un rôle fondamental: exprimées dès la conception de l'application, elles structurent l'exécution des processus et en déterminent la synchronisation. De plus, elles rendent la répartition transparente à l'exécution: toutes les communications sont uniformes quelles que soient les entités qui communiquent et quelles que soient leurs localisations respectives.

Cette communication est réalisée au moyen de messages échangés entre des portes qui constituent une interface logique de communication. Elle est intégrée au coeur du système d'exploitation sous-jacent à l'architecture.

Cet article présente en détail l'architecture CHORUS et son système exécutif. Il analyse et justifie les choix effectués. Un exemple simple et deux implantations illustrent cette architecture.

SUMMARY

CHORUS is an architecture for distributed systems, designed for a large class of machines and applications. The CHORUS architecture is built with a small set of powerful concepts. It is a solid and sound basis for building distributed applications, for learning distribution as well as for new researches and experiments.

CHORUS brings a new apprach to distribution where communications play a key-role: expressed in the design of a distributed application, they drive the process execution and induce synchronization. More, they render distribution transparent at run-time: all communications are uniform whatever be the entities which communicate and whatever be their respective locations.

That communication is acheived by the means of messages exchanged through ports which constitute the logical communication interface. Communication is integrated in the nucleus of the underlying operating system.

The paper presents in detail the CHORUS architecture and its operating system. Some aspects of CHORUS are then analyzed more deeply and discussed. Finally, a simple example and two implementations illustrate various aspects of the architecture.

TABLE DES MATIERES

TABLE DES MATIERES

TABLE DES MATIERES

1/ Introduction

CHORUS ([Banino 80], [Zimmermann 81], [Guillemont 82a]) est une architecture de système réparti conçue pour une grande variété de machines et de réseaux et une large classe d'applications: contrôle de processus industriel, productique, messagerie, bureautique, télématique, bases de données, télécommunications, systèmes de développement, etc... Elle comprend une méthode de conception d'applications réparties, une structure pour leur exécution et le système (d'exploitation) qui permet cette exécution.

Le projet CHORUS a été lancé en 1979. Cet article est une synthèse de CHORUS: résultats et perspectives. Les deux premières sections présentent l'architecture CHORUS et le système sous-jacent; la section 4 analyse et discute les options de CHORUS; la section 5 illustre CHORUS par un exemple; enfin, la section 6 présente deux implantations de CHORUS.

2/ L'architecture répartie CHORUS

Un système réparti CHORUS s'étend sur un ensemble de sites (c.a.d. de machines) interconnectés. Sur chaque site, les traitements sont réalisés par des entités actives appelées acteurs; le concept d'acteur est une adaptation à la répartition du concept traditionnel de processus séquentiel. Un acteur peut créer et détruire d'autres acteurs sur son site ou sur d'autres sites; il peut communiquer avec d'autres acteurs (locaux ou distants) au moyen de messages échangés à travers des portes. Acteurs, portes et messages peuvent être créés et détruits dynamiquement.

Sur chaque site, le système exécutif CHORUS permet et contrôle l'exécution des acteurs et leur offre des services systèmes. Sur chaque site, le système CHORUS comprend un noyau et un ensemble d'acteurs système.

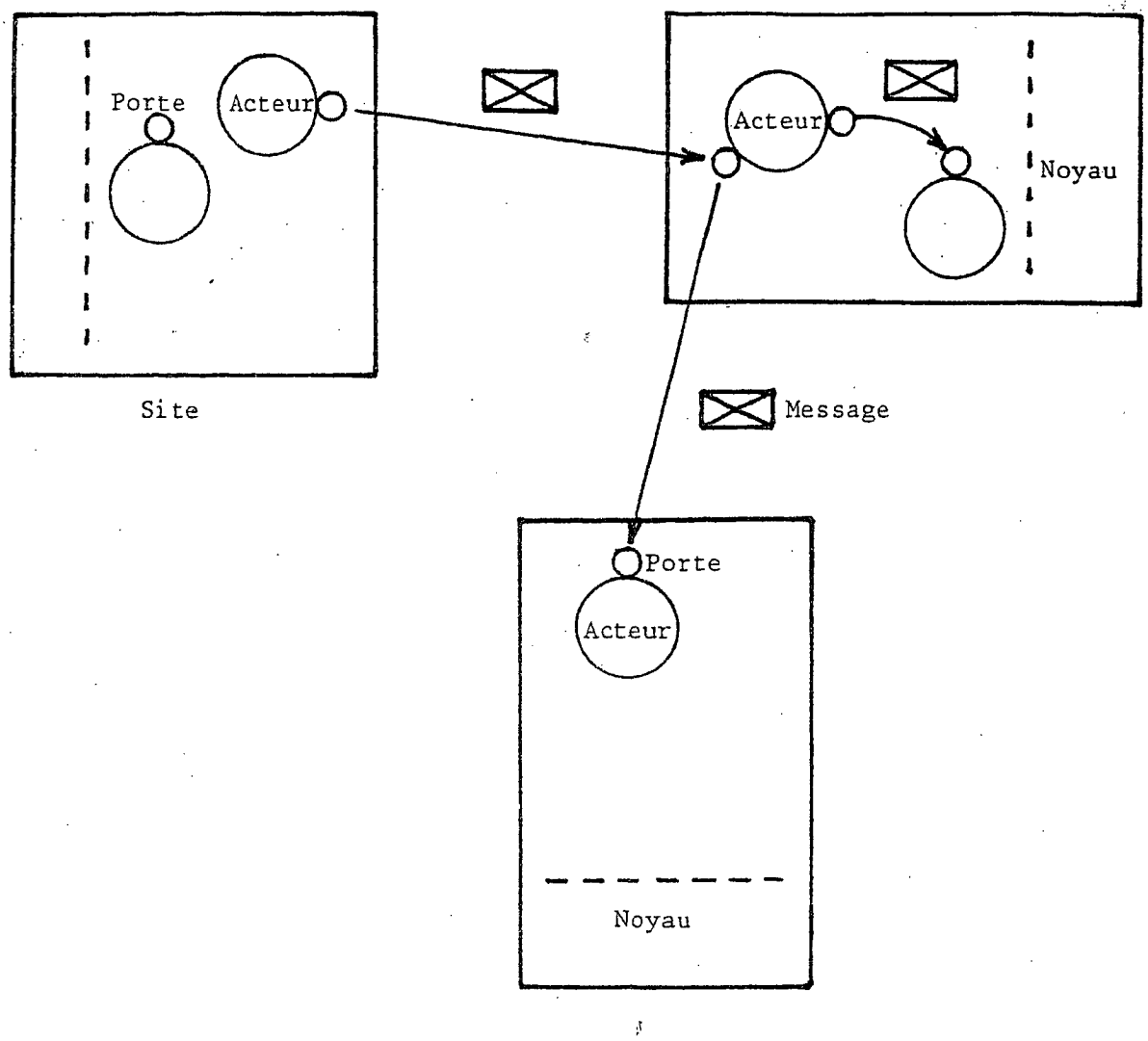

figure 2.1 : L'architecture répartie CHORUS

Le paragraphe 2.1 introduit les choix fondateurs de l'architecture CHORUS, tandis que les paragraphes 2.2 à 2.6 détaillent cette architecture.

2.1/ Les choix fondateurs

La définition de l'architecture CHORUS repose largement sur quatre choix fondamentaux qui concernent respectivement:

(a) La structure d'exécution d'une application répartie

(b) La structure d'exécution des acteurs

(c) L'unification des communications

(d) La standardisation des protocoles d'accès de service

2.1.1/ Structure d'exécution d'une application répartie

Une exécution répartie est souvent représentée par des processus communicants, c.a.d. par des processus classiques qui réalisent des opérations locales d'E/S pour envoyer et recevoir des messages. Cette approche de la répartition rend souvent difficile la description de protocoles où il est question d'échanges de messages (c.a.d. d'opérations réparties) plutôt que d'opérations locales d'envoi et de réception.

Dans CHORUS, le traitement et la communication jouent des rôles symétriques et complémentaires: l'exécution d'une application répartie est constituée d'étapes de communication (transferts de messages) et d'étapes de traitement (traitements de messages) qui s'enchaînent les unes les autres (voir figure 2.2). On peut donc considérer soit que les traitements déclenchent les communications (approche traditionnelle des traitements de données) soit que les communications déclenchent les traitements (approche traditionnelle des protocoles). Selon la nature du problème, une approche ou l'autre ou une combinaison des deux peut se révéler la mieux adaptée pour décrire et organiser des activités réparties.

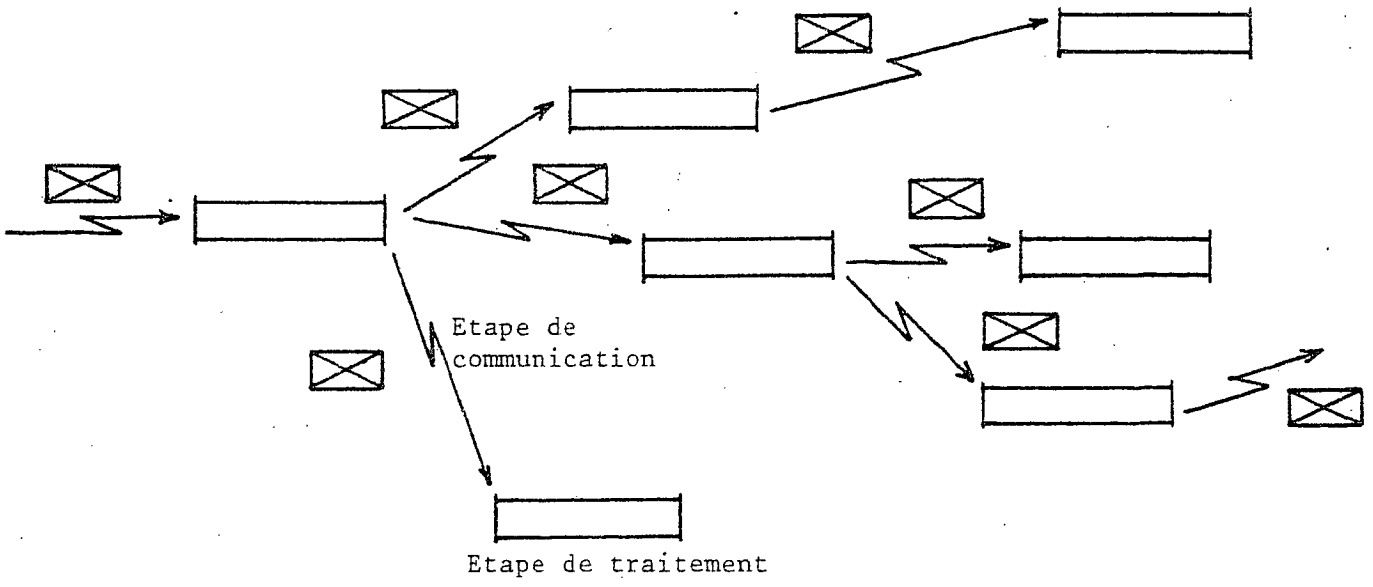

figure 2.2 : Execution d'une application répartie

2.1.2/ Structure d'exécution des acteurs

L'étape de traitement est l'unité de traitement réalisée par un acteur: elle est déclenchée par la réception d'un message et se termine par la transmission de n autres messages (n>=0).

Un acteur est l'entité active la plus élémentaire de CHORUS. Un acteur est purement local, c.a.d. entièrement - code et données - sur un site; il est purement séquentiel, c.a.d. qu'il n'y a pas entrelacement d'étapes de traitement au sein d'un même acteur (figure 2.3): les messages reçus sont traités un par un; chaque message reçu déclenche une étape de traitement et seule la réception d'un message peut déclencher une étape de traitement.

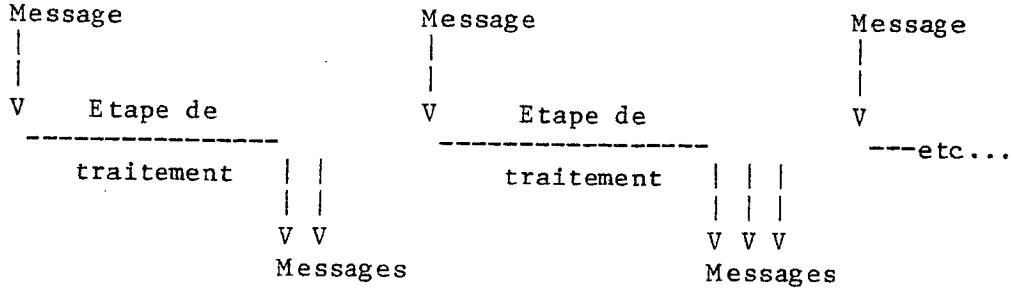

figure 2.3 : Exécution d'un acteur

2.1.3/ Unification des communications

Toutes les communications entre acteurs se font par échange de messages. Un message est envoyé d'une porte de l'acteur émetteur - la porte émettrice - vers une porte de l'acteur récepteur - la porte réceptrice - (ou vers plusieurs portes d'acteurs récepteurs). Les communications entre les acteurs et le système se font aussi par échange de messages entre des portes. Le service élémentaire de communication fourni par le système CHORUS est du type non-connecté [ISO 82] ; il est indépendant de la localisation, c.a.d. que les communications locales et distantes sont perçues de la même façon par les acteurs.

Subséquemment, un message sera représenté par le triplet

Porte émettrice, Porte réceptrice, [Texte du message]

2.1.4/ Standardisation des protocoles d'accès de service

Pour demander un service, un client doit communiquer avec le fournisseur du service. Dans CHORUS, cette communication prend la forme d'un échange de messages entre le client et le fournisseur selon un protocole d'accès de service.

Le protocole d'accès de service le plus courant est du type "Demande-Réponse", illustré par la figure 2.4, dans lequel

- la demande de service est un message envoyé de la porte Pc du client vers la porte Ps représentant le fournisseur du service,
- la réponse du service est un message envoyé de la porte Ps vers la porte Pc.

Demande: Pc, Ps, [Paramètres de demande]

Réponse: Ps, Pc, [Paramètres de réponse]

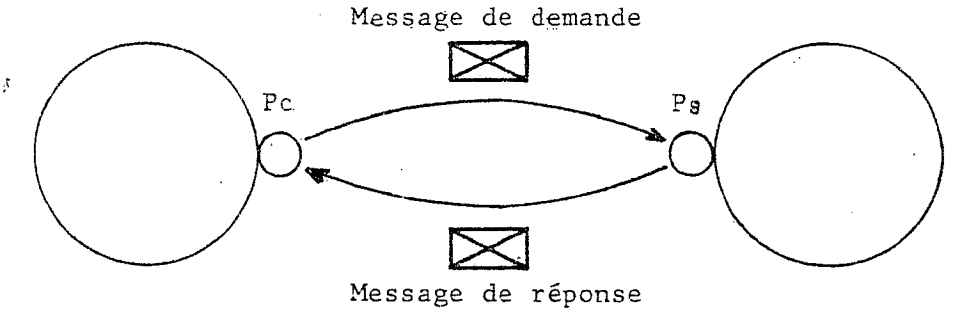

figure 2.4 : Protocole "Demande-Réponse" avec un service

Un autre protocole courant d'accès de service est du type "Ordre", illustré par la figure 2.5, dans lequel

- l'ordre du service est un message envoyé de la porte Pc du client vers la porte Ps représentant le fournisseur du service,
- aucune réponse n'est renvoyée par le service.

Ordre: Pc, Ps, [Paramètres d'ordre]

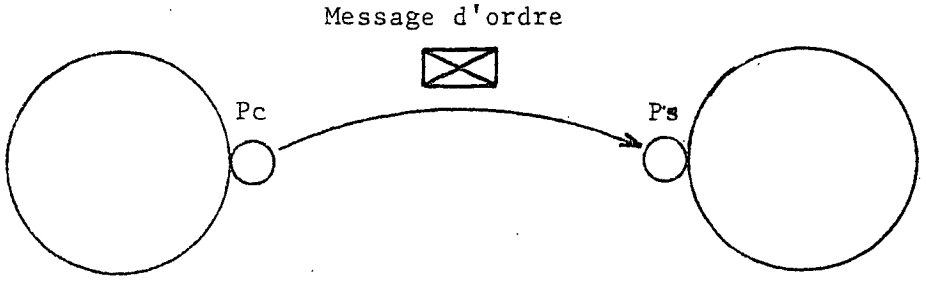

figure 2.5 : Protocole "Ordre" avec un service

Ces deux protocoles, Demande-Réponse et Ordre, sont les seuls utilisés pour l'accès aux services système dans CHORUS.

2.2/ Fonctionnement interne d'un acteur

Un acteur exécute une suite d'étapes de traitement. Chaque étape de traitement est précédée par deux opérations:

(a) une sélection (réalisée par le système) qui détermine quel est le prochain message traité par l'acteur, et

(b) un aiguillage (réalisé par le système) qui détermine quelle partie du code de l'acteur doit être exécutée dans cette étape de traitement.

Ces trois phases élémentaires de l'exécution d'un acteur (sélection, aiguillage, étape de traitement) sont décrites en détail dans les paragraphes 2.2.1 à 2.2.3.

2.2.1/ Le service de sélection

Les messages reçus par un acteur sont mis en file d'attente derrière leurs portes réceptrices respectives. A un instant donné, plusieurs messages peuvent être ainsi en attente derrière une ou plusieurs portes d'un acteur. Une sélection (réalisée par le système) permet de déterminer, à la fin de chaque étape de traitement, quel message sera traité par l'acteur au cours de l'étape

suivante.

Cette sélection peut être paramétrée dynamiquement par l'acteur en envoyant un message d'ordre au service système correspondant

Ordre au service:

Porte, Sélection, [liste de (Porte émettrice, Porte réceptrice)]

- Porte est le nom d'une porte de l'acteur à travers laquelle est envoyé l'ordre.
- Sélection désigne la porte représentant le service de sélection.
- Porte émettrice est le nom d'une porte.
- Porte réceptrice est le nom d'une porte de l'acteur à laquelle s'applique la sélection.

Effet: les messages reçus sur une des "Porte réceptrice" de la liste et envoyés par la "Porte émettrice" correspondante sont candidats pour la sélection, c.a.d. peuvent être traités par l'acteur au cours de la prochaine étape de traitement: chaque couple (Porte émettrice, Porte réceptrice) constitue une sorte de filtre pour sélectionner le prochain message à traiter et tous ces filtres fonctionnent en parallèle.

Conventions: le couple (Toutes, Porte réceptrice) permet à un acteur d'accepter tous les messages reçus sur sa "Porte réceptrice", tandis que le couple (Toutes, Toutes) permet à un acteur d'accepter tous les messages reçus sur toutes ses portes.

Note: ce service est accédé avec un protocole du type "Ordre", c.a.d. qu'il ne fournit aucune réponse; il enregistre simplement les paramètres de sélection de la liste. S'il y a une erreur dans cette liste, le couple erroné n'est pas

enregistré; si tous les couples sont erronés, la liste par défaut est (Toutes, Toutes).

Si plusieurs messages vérifient les conditions de sélection (c.a.d. passent chacun à travers un des filtres), le service de sélection tient compte des priorités des portes (cf paragraphe 2.3.4) et choisit un message reçu sur la porte de plus haute priorité. Enfin, si plusieurs messages remplissent ces deux conditions, le service de sélection choisit les messages selon leur ordre de réception.

Les conditions de sélection restent inchangées jusqu'à ce qu'un ordre de mise à jour soit transmis au système. Pour des raisons de simplicité, chaque ordre redéfinit la totalité des paramètres de sélection (il n'y a pas d'ordre de mise à jour partielle).

La valeur initiale des conditions de sélection est (Toutes, Toutes).

2.2.2/ Le service d'aiguillage

La partie du code qui doit être exécutée au cours d'une étape de traitement est désignée par son point d'entrée dans le code de l'acteur; chaque acteur possède donc plusieurs points d'entrée correspondant aux étapes de traitement qu'il est susceptible d'exécuter. Un aiguillage (réalisé par le système) permet de déterminer quelle partie de code doit être exécutée pour traiter le message qui vient d'être sélectionné par le service de sélection, c.a.d. à quel point d'entrée doit commencer l'exécution de l'étape de traitement.

Cet aiguillage peut être paramétré dynamiquement par l'acteur en envoyant un message d'ordre au service système correspondant

Ordre au service:

Porte, Aiguillage, [Point d'entrée]

- Porte est le nom d'une porte de l'acteur à laquelle s'applique le service d'aiguillage.
- Aiguillage désigne la porte qui représente le service d'aiguillage.
- Point d'entrée désigne un point d'entrée dans le code de l'acteur.

Effet: un message reçu sur "Porte" et sélectionné par le service de sélection déclenche l'exécution d'une étape de traitement désignée par "Point d'entrée".

Chaque ordre d'aiguillage ne modifie que le point d'entrée associé à "Porte", tandis que les points d'entrée associés aux autres portes restent inchangés.

La valeur initiale des paramètres d'aiguillage est l'association (Porte ombilicale, Point d'entrée initial) (cf paragraphe 2.4).

2.2.3/ L'étape de traitement

Quand un message a été sélectionné et aiguillé, l'étape de traitement est déclenchée dans l'acteur. Durant son exécution, l'acteur a accès au message qui a déclenché l'étape de traitement et à celui-là seulement; il a accès également à ses données internes. L'étape de traitement se termine par l'exécution de la primitive RETOURNER qui rend le contrôle au système et déclenche la transmission de tous les messages préparés pendant l'étape de traitement.

La programmation d'une étape de traitement peut donc se représenter de la façon suivante:

```
POINT-D'ENTREE e;
.....
{ traitement du message}
.....
RETOURNER (Pel, Prl, [Texte 1] ;
                .....
                Pen, Prn, [Texte n] );
```

figure 2.6 : Programmation d'une étape de traitement

Les "Pei, Pri, [Texte i]" sont les messages envoyés lorsque se termine l'étape de traitement; les portes Pei sont des portes de l'acteur.

Note: dans ce document, tous les exemples sont représentés de cette façon qui n'est qu'illustrative. La question d'un langage de programmation pour les acteurs est discutée au paragraphe 4.6.

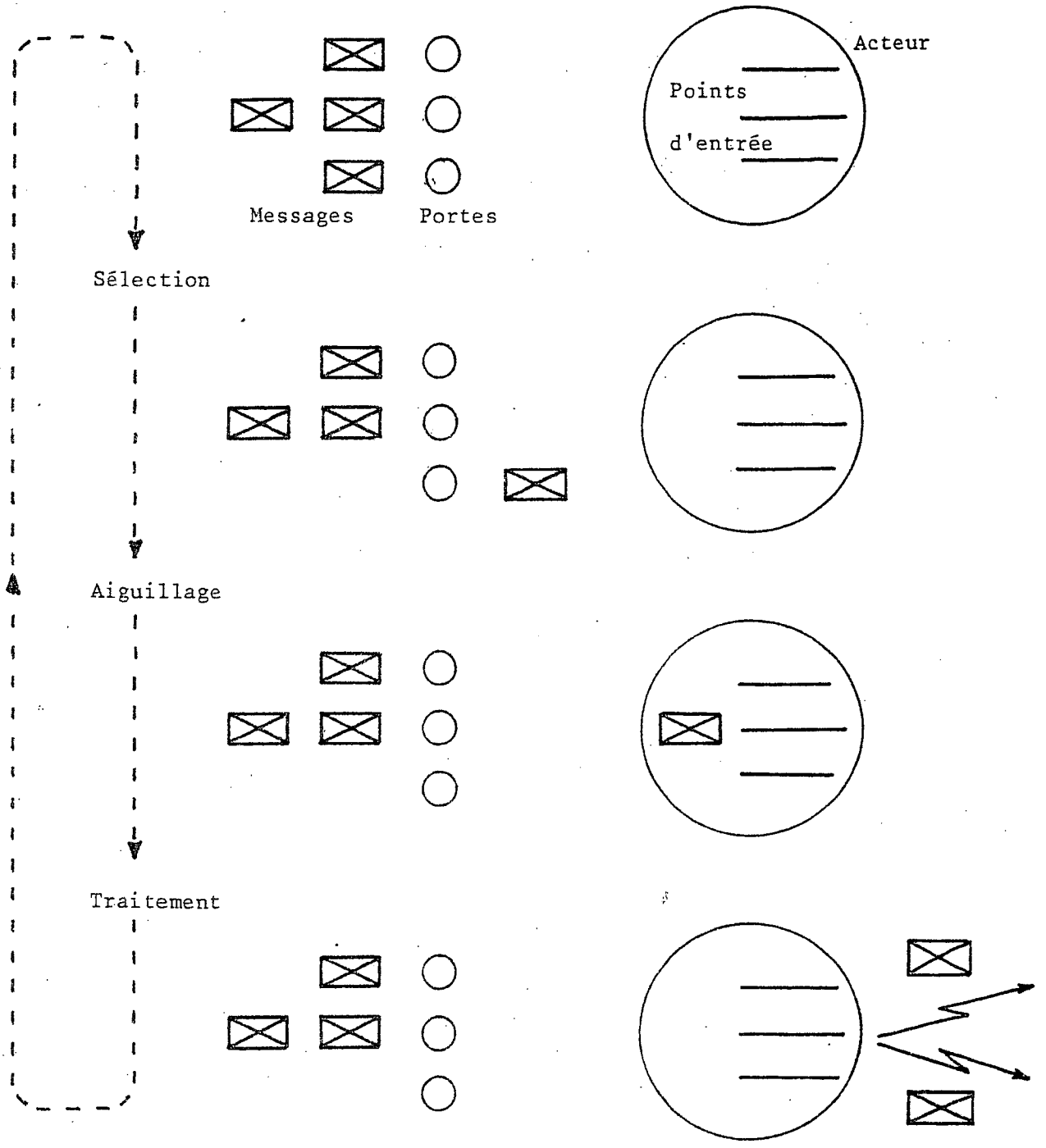

figure 2.7 : Exécution d'un acteur

2.3/ Les services de communication

Un acteur peut communiquer avec le monde extérieur (c.a.d. les autres acteurs et le système) ou avec lui-même en envoyant et en recevant des messages à travers des portes. Le service de transport des messages et le service de temporisation qui lui est associé sont décrits aux paragraphes 2.3.1 et 2.3.2.

Avant de pouvoir être utilisée pour envoyer et recevoir des messages, une porte doit d'abord exister puis être ouverte et liée à un acteur. Le système offre les services correspondants pour la gestion des portes: les services de création et de destruction de porte sont décrits au paragraphe 2.3.3 et les services d'ouverture et de fermeture de porte sont décrits au paragraphe 2.3.4.

2.3.1/ Le service de transport des messages

Le service de transport des messages permet aux acteurs d'envoyer et de recevoir des messages à travers des portes. Chaque porte peut être utilisée à la fois pour envoyer et pour recevoir des messages: les portes sont bidirectionnelles. Les messages sont transportés d'une porte émettrice vers une (ou plusieurs) porte(s) réceptrice(s).

Le service de transport des messages rend la répartition transparente aux acteurs en ce sens qu'ils n'ont pas à connaître la localisation de leur correspondants (c.a.d. sur quels sites ils se trouvent); ils n'ont besoin de connaître que les noms des portes de leurs correspondants et c'est à ces noms de portes qu'ils font référence pour envoyer des messages; le service de transport des messages se charge de localiser les portes réceptrices et d'acheminer les messages jusqu'à elles (cf paragraphe 3.6).

Le service élémentaire de communication est du type non-connecté temps réel. Aucune connexion n'est nécessaire pour envoyer ou recevoir des messages et les transports successifs de messages sont considérés (par le service de transport) comme des opérations indépendantes. Les messages sont transportés

aussi rapidement que possible; si un message ne peut pas être remis à son destinataire (porte réceptrice inaccessible ou inexistante ou fermée, etc...), le système le détruit. Enfin, le service ne fournit aucun compte-rendu sur le résultat, succès ou échec, du transport.

Quatre points sont à noter à propos de ce service de transport:

(a) L'aspect temps réel, non bloquant, de ce service est fondamental pour les applications temps réel.

(b) Des services de communication plus sophistiqués comme le transport sur connexion de l'ISO [ISO 82] ou le transport de masse sur liaison satellite de NADIR [Grangé 82] peuvent être construits au-dessus du service de transport des messages de CHORUS.

(c) Le transport local d'un message peut être aussi fiable, s'il est bien construit, que les communications traditionnelles de programme à programme (par appel de procédure, appel système ou file d'attente).

(d) CHORUS offre un service de temporisation qui est à la base des mécanismes de détection de perte de message (cf paragraphe 2.3.2).

Contrairement à tous les autres services, et pour d'évidentes raisons, ce service de transport des messages n'est pas appelé par un acteur au moyen d'un envoi de message. Ce service est invoqué automatiquement à la fin de chaque étape de traitement lorsque l'acteur exécute la primitive RETOURNER (cf paragraphe 2.2.3): tous les messages mentionnés dans cette primitive sont pris en charge par le service de transport des messages à ce moment-là.

Chaque message contient les informations suivantes:

Porte émettrice, Porte réceptrice, [Texte du message]

- Porte émettrice est le nom d'une porte (de l'acteur émetteur) à travers laquelle le message est émis.
- Porte réceptrice est le nom de la porte qui recevra le message.

La validité des messages envoyés et reçus peut être contrôlée par les procédures de contrôle d'émission et de réception (cf paragraphe 2.5).

Le fonctionnement interne du service de transport des messages est discuté au paragraphe 3.6.

2.3.2/ Le service de temporisation

Puisqu'une étape de traitement ne peut être déclenchée que par la réception d'un message, un acteur peut se trouver définitivement bloqué en attente d'un message "perdu" (par exemple parce que l'émetteur de ce message est tombé en panne avant l'émission ou parce que la transmission n'était pas possible). Afin d'éviter de tels blocages, un acteur peut armer une temporisation sur une "liaison" (c.a.d. un couple "Porte émettrice", "Porte réceptrice") en envoyant un message de demande au service système correspondant

Demande au service:

Porte, Temporisation, [Porte émettrice, Porte réceptrice, Délai]

- Porte est le nom d'une porte de l'acteur.
- Temporisation désigne la porte qui représente le service de temporisation.
- Porte émettrice est le nom d'une porte.
- Porte réceptrice est le nom d'une porte de l'acteur.
- Délai est un délai.

Effet:

- si aucun message émis par la "Porte émettrice" n'est reçu sur la "Porte réceptrice" avant l'expiration du "Délai", le système génère un message diagnostique (conformément au protocole "Demande-Réponse")

Temporisation, Porte, [Porte émettrice, Porte réceptrice]

Ce message diagnostique, comme n'importe quel autre message, déclenche une étape de traitement dans l'acteur. Ce message est "filtré" par le service de sélection comme un message émis par la "Porte émettrice" et reçu sur la "Porte réceptrice".
- si un message émis par la "Porte émettrice" est reçu sur la "Porte réceptrice" avant l'expiration du "Délai", la temporisation est désarmée.

Convention: la convention "[Toutes, Porte réceptrice Délai]" permet à un acteur d'armer une temporisation qui sera désarmée par n'importe quel message reçu sur "Porte réceptrice".

Note: plusieurs temporisations avec différentes "Porte émettrice" peuvent être armées simultanément sur la même porte.

En d'autres termes, ce service réalise une surveillance de la liaison ("Porte émettrice", "Porte réceptrice") et le résultat de cette surveillance (en cas d'erreur) est un message reçu sur "Porte".

2.3.3/ Les services de création et de destruction de porte

Les portes sont créées à partir de modèles de porte qui définissent en particulier les procédures de contrôle d'émission et de réception associées aux portes (cf paragraphe 2.5). Plusieurs portes peuvent être créées à partir du même modèle. A sa création, une porte reçoit un nom global unique qui sera ensuite utilisé pour la désigner.

Un acteur peut demander la création d'une porte en envoyant un message de demande au service système correspondant

Demande au service:

Porte, Creation_de_Porte, [Modèle, Nom, Paramètres d'initialisation]

- Porte est une porte de l'acteur.
- Creation_de_Porte désigne la porte qui représente le service de création de porte.
- Modèle est le nom du modèle à partir duquel la porte doit être créée.
- Nom est le nom global unique de la porte; si ce paramètre n'est pas fourni, la porte reçoit un nouveau nom global unique choisi par le système.
- Les paramètres d'initialisation sont utilisés pour initialiser les procédures de contrôle associées à la porte.

Réponse du service:

Creation_de_Porte, Porte, [Diagnostic, Nom de la porte créée]

De même qu'un acteur peut demander la création d'une porte, il peut aussi demander la destruction d'une porte en envoyant un message de demande au service système correspondant

Demande au service:

Porte, Destruction_de_Porte, [Nom]

- Nom est le nom de la porte à détruire.

Réponse du service:

Destruction_de_Porte, Porte, [Diagnostic]

Remarque: pour que la destruction d'une porte soit possible, il faut que cette porte soit fermée (cf paragraphe 2.3.4); sinon, la destruction est refusée.

La création et la destruction des portes sont protégées par les procédures de contrôle de création et de destruction (cf paragraphe 2.5) associées à la porte.

2.3.4/ Les services d'ouverture et de fermeture de porte

Avant de pouvoir utiliser une porte (pour recevoir et/ou émettre des messages), un acteur doit l'ouvrir; l'ouverture d'une porte est analogue à l'établissement d'un lien entre la porte et l'acteur. Quand un acteur n'a plus besoin d'une porte, il la ferme (c.a.d. qu'il se "délie" de la porte). Une porte ne peut être ouverte que par un seul acteur à la fois (c.a.d. liée à un seul acteur à la fois), mais la même porte peut être ouverte puis fermée successivement par différents acteurs.

Les messages envoyés sur une porte qui n'est pas ouverte (c.a.d. qui n'est pas "liée" à un acteur) sont perdus.

Un acteur peut demander l'ouverture d'une porte en envoyant un message de demande au service système correspondant

Demande au service:

Porte, Ouverture_de_Porte, [Nom, Priorité]

- Nom est le nom de la porte à ouvrir
- Priorité est la priorité allouée à la porte et utilisée pour le cadencement de acteurs (cf paragraphe 3.1.1).

Réponse du service:

Ouverture_de_Porte, Porte, [Diagnostic]

L'ouverture d'une porte est protégée par une procédure de contrôle d'ouverture (cf paragraphe 2.5). Ce contrôle peut être utilisé, par exemple, pour empêcher un acteur d'ouvrir une porte représentant un service qu'il ne sait pas offrir.

Réciproquement, un acteur peut demander la fermeture d'une porte, en envoyant un message de demande au service système correspondant

Demande au service:

Porte, Fermeture_de_Porte, [Nom]

- Nom est le nom de la porte à fermer, qui doit être liée à l'acteur qui demande sa fermeture.

Réponse du service:

Fermeture_de_Porte, Porte , [Diagnostic]

Remarque: un acteur ne peut fermer que les portes qu'il a ouvertes.

Il n'y a pas de procédure de contrôle de fermeture de porte car un acteur peut toujours fermer les portes qu'il a ouvertes, sauf sa porte ombilicale (cf paragraphe 2.4).

2.4/ Les services de création et de destruction d'acteur

Un acteur est créé à partir d'un modèle d'acteur qui définit en particulier son code et ses données (cf paragraphe 2.6). Plusieurs acteurs peuvent être créés à partir du même modèle.

Un acteur est créé avec une porte ombilicale sur laquelle il reçoit un message initial; la porte ombilicale est automatiquement aiguillée sur le point d'entrée initial; subséquemment, le message initial déclenche l'exécution d'une étape de traitement initiale.

Un acteur peut demander la création d'autres acteurs, localement ou sur un autre site, en faisant appel au service de création d'acteur. Un acteur peut demander sa propre destruction ou celle d'autres acteurs en faisant appel au service de destruction d'acteur. Les messages de demande et de réponse échangés avec ces services sont similaires à ceux échangés avec les services analogues sur les portes (cf paragraphe 2.3.3).

La création et la destruction des acteurs sont protégées par les procédures de contrôle de création et de destruction d'acteur (cf paragraphe 2.5). Ces contrôles peuvent, par exemple, empêcher un acteur de détruire inopportunément un autre acteur.

2.5/ Protection

Dans CHORUS, chaque acteur avec ses portes forme un domaine élémentaire pour la protection. CHORUS permet de contrôler dynamiquement la création des domaines et leurs interactions (cf paragraphes 2.3 et 2.4, où plusieurs procédures de contrôle ont été mentionnées); par contre CHORUS ne contrôle pas le fonctionnement interne des acteurs au cours d'une étape de traitement.

Les mécanismes de protection de CHORUS peuvent être divisés en deux classes:

- des contrôles standards qui imposent le respect de l'architecture et qui sont réalisés automatiquement par le système (par exemple "une porte ne peut être ouverte que par un seul acteur à la fois", ou "un acteur ne peut émettre un message qu'à travers une porte qu'il a ouverte", etc...).
- des contrôles spécifiques, définis par des procédures de contrôle, qui imposent le respect de règles définies par l'utilisateur; ces contrôles permettent d'adapter la protection à chaque classe d'application.

Il y a une procédure de contrôle pour chaque type de service significatif (création, destruction, ouverture, émission, etc...) associée à chaque acteur ou chaque porte; par exemple, chaque acteur possède une "procédure de contrôle de destruction", chaque porte possède une "procédure de contrôle d'émission", etc... Ces procédures de contrôle de CHORUS sont résumées dans le tableau ci-dessous.

Service système	Procédure de contrôle	Associée à
Création d'acteur	oui	modèle d'acteur
Destruction d'acteur	oui	acteur
Création de porte	oui	modèle de porte
Destruction de porte	oui	porte
Ouverture de porte	oui	porte
Fermeture de porte	non	
Emission de message	oui	porte
Reception de message	oui	porte

figure 2.8 : Les procédures de contrôle

Les procédures de contrôle sont exécutées par le noyau sur chaque site au moment de la réalisation du service système correspondant.

Par exemple, lorsqu'un acteur A demande la destruction d'un acteur B, le système exécute la "procédure de contrôle de destruction" associée à B; si le résultat est positif, la requête de destruction est considérée comme valide et exécutée; si le résultat est négatif, la requête est considérée comme invalide et non exécutée. Dans tous les cas, l'acteur A reçoit un diagnostic correspondant au résultat de sa requête dans le message de réponse du service.

Un autre usage important des procédures de contrôle concerne la protection des communications; par exemple, un acteur A envoie le message "Pa, Pb, [M]" de sa porte Pa vers la porte Pb (ouverte par un acteur B):

- lorsque le message est émis de la porte Pa, le système exécute la "procédure de contrôle d'émission" associée à Pa; cette procédure contrôle par exemple que M est cohérent au regard des spécifications de l'acteur A; si le contrôle est négatif, le message est détruit et le noyau génère éventuellement un message diagnostique (cf paragraphe 3.4).
- lorsque le message est reçu par la porte Pb, le système exécute la

"procédure de contrôle de réception" associée à Pb; cette procédure contrôle par exemple que l'acteur B peut traiter M; si le contrôle est négatif, le message est détruit et le noyau génère éventuellement un message diagnostique.

Ces procédures de contrôle sont définies par l'utilisateur et elles peuvent donc être étroitement adaptées à chaque application; elles peuvent aussi être modifiées (sans changement dans le code des acteurs) entre une phase de mise au point et une phase d'exploitation. Plus tard, avec l'expérience, des procédures de contrôle standards seront disponibles dans les bibliothèques système.

2.6/ Construction d'un acteur

Les acteurs sont créés à partir de modèles d'acteur (cf paragraphe 2.4). Tous les acteurs issus du même modèle ont le même code et les mêmes données initiales. Les paramètres d'initialisation spécifiques de chaque acteur sont transmis dans le message initial.

CHORUS permet de construire un modèle d'acteur en deux étapes:

(a) Une application répartie (ou une partie d'application) est décrite comme un ensemble de modules communiquant à travers des portes, comme des acteurs. Un module correspond à la description de plusieurs étapes de traitement qui travaillent sur des données communes.

(b) L'application (ou une partie d'application) est ensuite configurée en regroupant les modules en modèles d'acteur, chaque modèle comprenant un ou plusieurs modules ainsi que les procédures de contrôle (cf paragraphe 2.5).

Le regroupement de plusieurs modules dans un modèle d'acteur permet d'améliorer les performances (par exemple, la communication entre modules dans un même modèle d'acteur peut se faire par un appel de procédure au lieu d'un échange de messages). Cependant, ce regroupement impose que tous les modules soient exécutés sur la même machine et appartiennent au même domaine de protection.

- Une application répartie peut être reconfigurée (par exemple pour améliorer les performances, pour répartir la charge, etc...) en changeant la définition des modèles d'acteur sans changer sa description en termes de modules.

3/ Le système CHORUS

Le système CHORUS qui permet l'exécution de l'architecture CHORUS est constitué d'un noyau (résident sur chaque site) et d'acteurs système (qui n'ont pas besoin d'être tous présents sur chaque site). Cette section présente l'organisation du système CHORUS, y inclus le traitement des interruptions et des E/S qui ont été intégrés dans l'architecture CHORUS.

3.1/ Structure du système

3.1.1/ Le noyau

Présent sur chaque site, le noyau permet l'exécution des acteurs locaux en réalisant les fonctions suivantes:

- gestion des conditions de sélection et cadencement des acteurs,
- gestion des conditions d'aiguillage,
- gestion des temporisations,
- transport local des messages.

De plus, le noyau participe à

- la gestion des interruptions (cf paragraphe 3.2),
- la réalisation des E/S (cf paragraphe 3.3).

Le fonctionnement du noyau peut être vu comme une séquence cyclique de trois phases: sélection, aiguillage et retour (cf ci-dessous). Il est à noter que du fait des interruptions, des priorités et du multitraitement, le noyau peut gérer plusieurs cycles de cette sorte en parallèle.

Phase de sélection

Le noyau détermine le prochain message à traiter par un acteur local; ce message M, reçu sur la porte P d'un acteur A, est choisi parmi tous les messages reçus sur le site selon les critères suivants:

(a) L'acteur A ne doit pas être en cours d'exécution d'une étape de traitement (puisqu'il ne doit pas y avoir entrelacement d'étapes de traitement au sein d'un même acteur),

(b) Le message M doit vérifier les conditions de sélection définies par l'acteur A pour la porte P (cf paragraphe 2.2.1),

(c) Si plusieurs messages vérifient les conditions (a) et (b), la porte P choisie est celle qui a la plus haute priorité (cf paragraphe 2.3.4),

(d) Enfin, si plusieurs messages vérifient les conditions (a), (b) et (c), le message M choisi est celui qui a été reçu le premier.

Phase d'aiguillage

Lorsque le message M a été sélectionné, l'acteur A devient activable. Le noyau détermine le point d'entrée associé à la porte P (conformément à la condition d'aiguillage donnée par A pour la porte P, cf paragraphe 2.2.2), installe le contexte associé et lance l'exécution de l'étape de traitement.

Phase retour

Lorsque l'étape de traitement se termine (avec la primitive RETOURNER), le noyau examine tous les messages que lui transmet l'acteur et:

(a) il traite les messages envoyés sur une de ses propres portes (ces messages correspondent à des demandes de services réalisés par le noyau, cf paragraphe 3.1.4),

(b) il transporte les autres messages sur des portes locales (cf paragraphe 3.6).

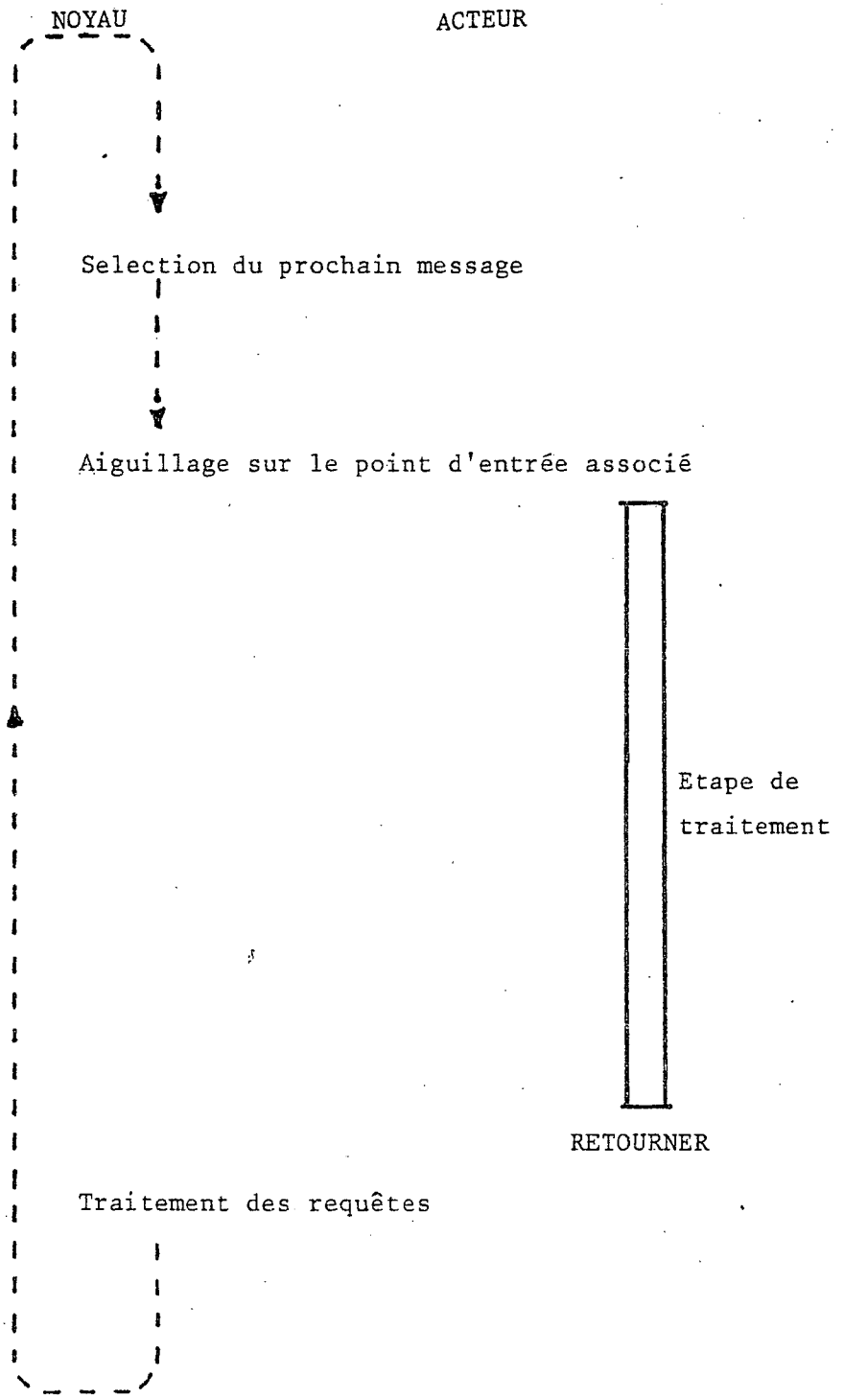

figure 3.1 : Fonctionnement schématique du noyau

3.1.2/ Les acteurs système

Les services système qui ne sont pas réalisés par le noyau le sont par des acteurs système, en particulier:

- création et destruction des acteurs,
- création, destruction, ouverture et fermeture des portes,
- communication distante (cf paragraphe 3.6),
- gestion de la mémoire,
- gestion des noms,
- gestion des consoles,
- gestion des fichiers,
- etc...

Les acteurs système ne diffèrent des acteurs d'application que par le fait qu'ils ont certains droits spécifiques et qu'ils peuvent avoir des communications priviligiées avec le noyau (par messages), par exemple pour lui signaler des modifications sur les acteurs et les portes (cf paragraphe 3.1.3) ou pour demander des E/S (cf paragraphe 3.3).

3.1.3/ Relations noyau / acteurs système

Pour réaliser ses fonctions, le noyau a besoin d'informations telles que les noms des acteurs et des portes locaux, les associations entre les acteurs et les portes, etc... Ces informations sont regroupées en deux tables:

(a) la table des acteurs locaux, qui contient pour chaque acteur: le nom de son modèle, son nom, son état d'exécution, son contexte d'exécution, les noms de ses portes, les conditions de sélection.

(b) la table des portes locales ouvertes (et celles-là seulement), qui contient pour chaque porte: son nom, sa priorité, le nom de l'acteur qui l'a ouverte, la liste des messages reçus, les temporisations en cours (éventuellement), les conditions d'aiguillage.

La réalisation d'un service par un acteur système peut conduire à des modifications des tables du noyau; par exemple, si un acteur ouvre une porte P, cette porte P doit être insérée dans la table des portes locales ouvertes. Pour d'évidentes raisons de modularité et de protection, les acteurs système n'accèdent pas directement aux tables du noyau; au contraire, ils transmettent au noyau des messages tels que "la porte P est ouverte" et le noyau met à jour ses tables. Par conséquent, les acteurs système n'ont pas à connaître la structure des tables du noyau.

Cette séparation entre le noyau et les acteurs système présente plusieurs avantages:

- l'interface du noyau est clairement définie par la liste de ses portes et des messages qu'il peut recevoir et envoyer,
- l'organisatin des tables du noyau peut évoluer sans aucune répercussion sur les acteurs système,
- le code des acteurs système est moins dépendant de la machine et donc plus facilement portable.

3.1.4/ L'interface du système

Pour des raisons d'homogénéité, le système est vu par les acteurs comme un ensemble de portes: le noyau et les acteurs système reçoivent et envoient des messages à travers des portes. L'accès à un service système suit toujours l'un des deux protocoles d'accès de service décrits au paragraphe 2.1.4 (Demande-Réponse ou Ordre). Ceci offre plusieurs avantages:

- un acteur n'a pas à connaître la répartition des services entre le noyau et les acteurs système,
- la répartition des services entre le noyau et les acteurs système peut être modifiée sans affecter les acteurs d'application,
- certains acteurs système peuvent être distants sans que cela soit visible des acteurs utilisateurs puisque la communication est indépendante de la

localisation des portes. (La seule contrainte est que les acteurs système qui transmettent des informations à un noyau doivent être sur le même site que ce noyau.)

Par exemple, dans l'implantation actuelle de CHORUS, les portes du noyau sont: Selection, Aiguillage, Temporisation (cf paragraphes 2.2.1, 2.2.2 et 2.3.2) et quelques autres pour les communications avec les acteurs système.

Les autres portes, comme Creation_de_Porte, Destruction_de_Porte, Ouverture_de_Porte, etc... (cf paragraphes 2.3.3 et 2.3.4) sont des portes d'acteurs système.

Sur chaque site, le système minimum est constitué d'un noyau et d'un petit ensemble d'acteurs système:

- un "gestionnaire d'acteurs",
- un "gestionnaire de portes",
- une "station de transport",
- un "gestionnaire de consoles" sur les sites qui ont une console,
- un "gestionnaire de fichiers".

Les autres acteurs système peuvent être distants.

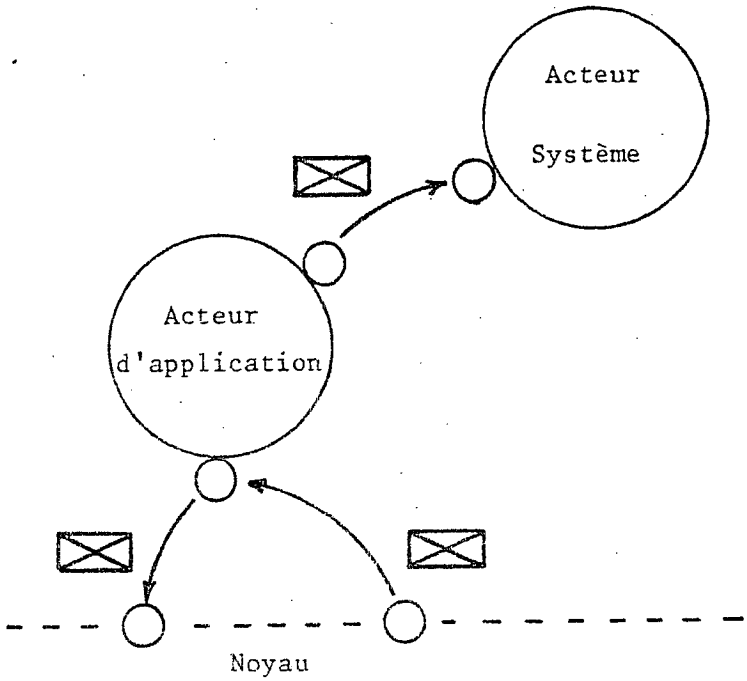

figure 3.2 : L'interface du système

3.2/ Les interruptions

Les interruptions ne déclenchent pas directement l'exécution d'un acteur: ceci ne serait pas conforme au mécanisme général selon lequel seul un message peut déclencher l'exécution d'une étape de traitement. Au contraire, les interruptions sont perçues par les acteurs comme des messages d'interruption envoyés par des portes d'interruption selon le principe suivant:

Un acteur système AGI (Acteur Gestionnaire des Interruptions) ouvre une porte Pi pour chaque niveau d'interruption i accessible aux acteurs; cette porte Pi représente le mécanisme d'interruption de niveau i pour les acteurs. Quand un acteur A veut traiter les interruptions du niveau i, c.a.d. recevoir les messages d'interruption de niveau i, il envoie un message de demande sur la porte Pi:

Pa, Pi, [association au niveau d'interruption]

L'acteur AGI reçoit ce message et vérifie la validité de la demande; si le contrôle est positif, l'AGI envoie au noyau le triplet [i, Pi, Pa] :

Ps, Pn, [i, Pi, Pa]

Le noyau enregistre ce triplet dans ses tables et arme le niveau d'interruption i.

Quand une interruption de niveau i se produit, le noyau la reçoit et il crée un message d'interruption qu'il envoie sur Pa:

Pi, Pa, [une interruption s'est produite]

Ce message déclenche une étape de traitement selon les mêmes règles que les autres messages.

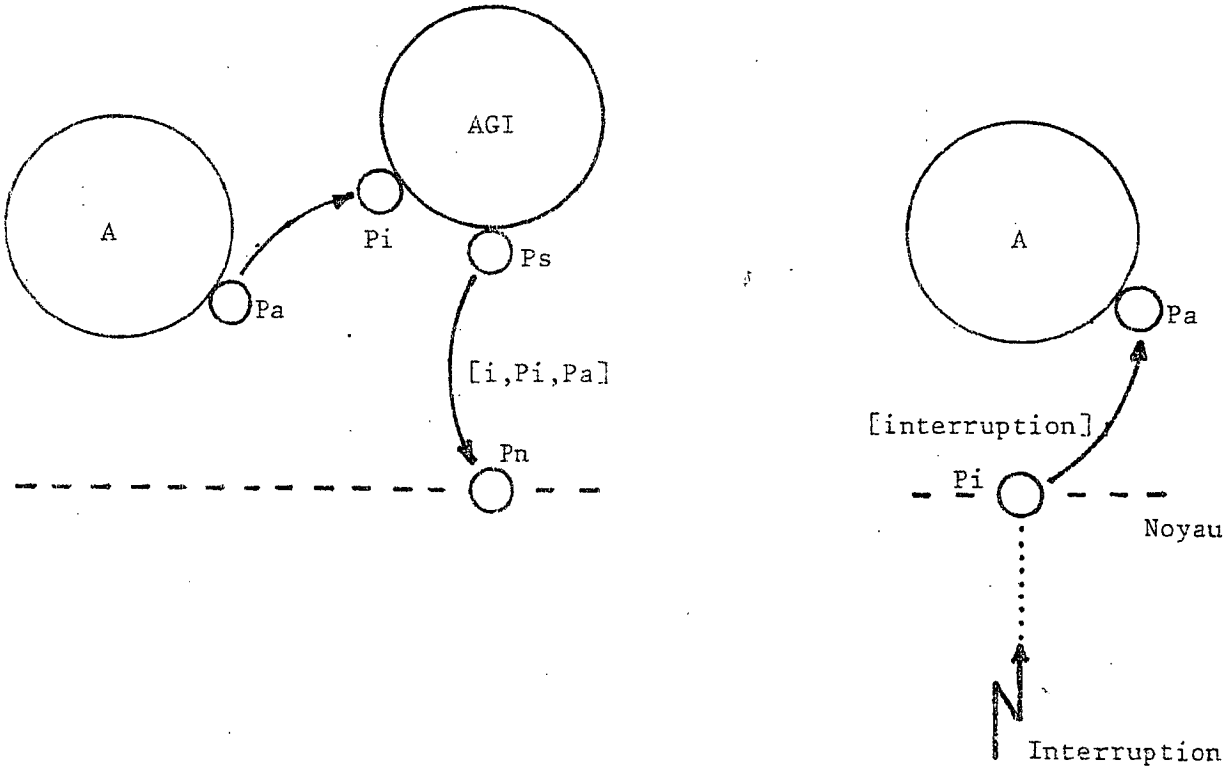

figure 3.3 : Implantation des interruptions

Selon les caractéristiques de chaque site, la réquisition peut être autorisée ou non. Si oui, lorsque le noyau a transformé l'interruption en un message d'interruption déposé sur la porte Pa, l'acteur actif peut être suspendu

et l'étape de traitement associée à la porte Pa peut être déclenchée (cf ci-dessous). La réquisition améliore le temps de réponse aux interruptions prioritaires et elle est donc généralement mise en oeuvre dans les systèmes ayant de fortes contraintes de temps réel. Ce mécanisme doit cependant respecter deux conditions:

1/ l'acteur interrompu ne peut pas être l'acteur qui traite l'interruption: il n'y a pas de réquisition ou de parallélisme entre étapes de traitement à l'intérieur d'un même acteur, c.a.d. qu'un acteur ne peut pas être interrompu au cours d'une étape de traitement pour traiter un autre message,

2/ les priorités des étapes de traitement (c.a.d. les priorités des portes associées) doivent être respectées.

3.3/ Les Entrées / Sorties

Comme dans les systèmes classiques, les acteurs système offrent aux acteurs d'application une interface de haut niveau pour l'accès aux périphériques.

Pour les acteurs système chargés du traitement des E/S, les périphériques physiques sont vus comme des portes: une demande d'E/S est un message envoyé sur cette porte; une fin d'E/S est un message envoyé par cette porte.

Par exemple, un périphérique physique D est représenté par une porte Pd (du noyau). Un acteur système AGD (Acteur Gestionnaire du périphérique D) chargé de gérer D demande une E/S élémentaire sur D en envoyant le message

Ps, Pd, [Demande d'une E/S]

Ce message contient les paramètres de l'E/S (lecture ou écriture, nombre d'octets, etc...). Le noyau reconnait Pd comme représentant le périphérique D et il déclenche l'E/S élémentaire; la fin d'E/S correspond à une interruption, transformée (par le noyau) en un message (cf paragraphe 3.2)

Pd, Ps, [Fin d'E/S]

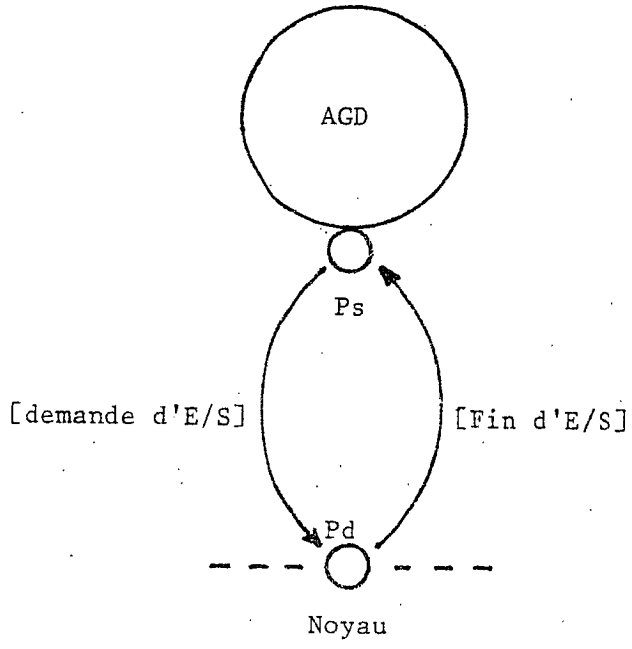

figure 3.4 : Implantation des E/S

Pour les acteurs système, les opérations d'E/S disponibles sont celles offertes par le périphérique lui-même: par exemple, E/S d'un caractère sur une ligne asynchrone, E/S de 128 octets sur une disquette, etc...

Cette organisation des E/S permet une implantation plus aisée de CHORUS sur des machines multi-processeurs où les E/S sont réalisées par des processeurs spécialisés: un processeur d'E/S peut facilement être adapté pour recevoir et envoyer des messages d'une porte Pd; évidemment, ceci ne concerne que l'interface du processeur et n'implique pas du tout que ce processeur supporte l'architecture CHORUS.

3.4/ Traitement des erreurs

Diverses erreurs peuvent se produire pendant l'exécution d'un acteur:

- erreur dans une demande de service (paramètre incorrect, demande non autorisée, etc...),
- erreur de comportement (envoi d'un message incorrect, etc...),
- erreur dans une instruction (division par zéro, erreur d'adressage, etc...).

CHORUS permet aux acteurs de récupérer certaines de ces erreurs selon le schéma suivant:

Les erreurs sont détectées par les acteurs système, le noyau ou la machine (selon l'erreur); dans les deux premiers cas, l'acteur système ou le noyau peuvent envoyer un message diagnostique sur une porte spécifiée par l'acteur qui a commis l'erreur; dans le dernier cas, le noyau reçoit l'erreur et la transforme en un message. Dans tous les cas, les acteurs reçoivent les signalements d'erreur sous forme de messages. La porte qui reçoit le message diagnostique n'appartient pas nécessairement à l'acteur qui a produit l'erreur; par exemple, un acteur A peut surveiller un acteur B et recevoir tous les messages diagnostiques consécutifs aux erreurs de B.

Cependant, si un acteur ne veut pas récupérer ses erreurs, il est détruit (en cas d'erreur, bien sûr!...) et un message diagnostique est envoyé à un autre acteur.

3.5/ Désignation

Dans CHORUS, chaque entité (acteur ou porte) est désignée par un nom global qui désigne cette entité et celle-là seulement; cette désignation est constante dans l'espace et dans le temps, en ce sens qu'un nom global désigne toujours la même entité quel que soit le site sur lequel se trouve l'acteur qui utilise ce nom et quel que soit le moment où il l'utilise. Un nom global est créé pour chaque nouvelle entité et il n'est jamais réutilisé pour désigner une autre

entité.

Un nom global est obtenu par la concaténation de trois champs:

site | type de nom | estampille locale unique

"Site" désigne le site de création de l'entité; comme tous les noms de site sont distincts, la concaténation d'un nom de site et d'une estampille locale unique produit un nom global unique. Notons bien que "site" désigne le site de création de l'entité et non son site de résidence: une entité peut changer de site sans changer de nom.

"Type de nom" permet de distinguer plusieurs classes d'entités. Par exemple, "acteur" ou "porte"; ou encore "sédentaire" ou "nomade": une entité nomade est une entité qui peut changer de site alors qu'une entité sédentaire ne le peut pas (cet attribut est défini à la création de l'entité et ne peut pas varier); cette distinction facilite la recherche du site de résidence d'une entité: une entité sédentaire se trouve nécessairement sur son site de création.

De façon élémentaire, tous les mécanismes de désignation de CHORUS utilisent les noms globaux: par exemple, dans les communications, un acteur désigne les portes émettrice et réceptrice d'un message par leurs noms globaux; un service est désigné par le nom global de la porte qui l'offre; si un acteur veut détruire un autre acteur, il le désigne par son nom global; etc... Cependant, pour des raisons de performance, des accélérateurs peuvent être construits au-dessus de ce mécanisme pour obtenir une désignation plus rapide et moins coûteuse (cf paragraphe 4.2).

Cette grande portée des noms globaux est tempérée par les mécanismes de protection (cf paragraphe 2.5): un acteur peut demander la réalisation de n'importe quel service sur n'importe quelle entité, mais ce service ne sera réalisé que si les mécanismes de protection le permettent (la communication est un service demandé sur les portes).

3.6/ Les communications locales et distantes

Le système CHORUS offre un service de communication entre portes qui est indépendant des leurs localisations (cf paragraphe 2.3.1); le transport local des messages est réalisé par le noyau (cf paragraphe 3.1.1). Ce paragraphe présente le mécanisme de transport complet, c.a.d. local et distant, dans le mode datagramme de base.

Soit un acteur A qui envoie le message M "Pa, Pb ,[Texte du message]". Le noyau reçoit ce message à la fin d'une étape de traitement de A. (Le noyau a une table des portes locales ouvertes, cf paragraphe 3.1.3.)

(1) Si Pb se trouve dans cette table des portes locales ouvertes, le noyau réalise le transport local du message: il place M dans la file d'attente de Pb.

(2) Sinon (c.a.d. si Pb est ouverte et distante ou fermée ou inexistante), le noyau place M dans la file d'attente de la porte Pst qui est la porte locale par défaut, représentant toutes les portes distantes; cette porte Pst est une porte de l'acteur local Station de Transport (AST1). M déclenche une étape de traitement de l'acteur AST1. Cet acteur AST1 applique un algorithme de routage (décrit ci-dessous) pour trouver le site de résidence de Pb; si cet algorithme aboutit (c.a.d. si Pb est ouverte, distante et située sur un site accessible), AST1 transmet M, à travers un support de communication, vers AST2 qui est l'acteur Station de Transport du site de résidence de Pb; si cet algorithme n'aboutit pas, le message M est détruit. AST2 envoie alors "Pa, Pb, [Texte du message]" et le noyau local place M dans la file d'attente de Pb.

Remarque 1: AST1 reçoit M sur sa porte Pst qui n'est pas la porte réceptrice de M et AST2 envoie M au nom de Pa qui n'est pas une de ses portes ouvertes. Ces deux "dérogations" sont nécessaires pour conserver une vision uniforme des communications qu'elles soient locales ou distantes: quelles que soient les localisations respectives de Pa et Pb, l'acteur A envoie M de Pa vers

Pb et l'acteur B reçoit M envoyé par Pa et reçu sur Pb. Le noyau contrôle strictement l'usage de ces deux dérogations dans la mesure où il sait que AST est l'acteur Station de Transport: seul cet acteur peut faire usage de ces dérogations.

Remarque 2: si Pb n'est pas accessible (c.a.d. fermée, inexistante ou située sur un site inaccessible), le message M est perdu et le service de transport ne fournit aucun diagnostic; la détection et le traitement de ce genre d'erreur est du ressort de protocoles de transport construits au-dessus de ce mécanisme de base.

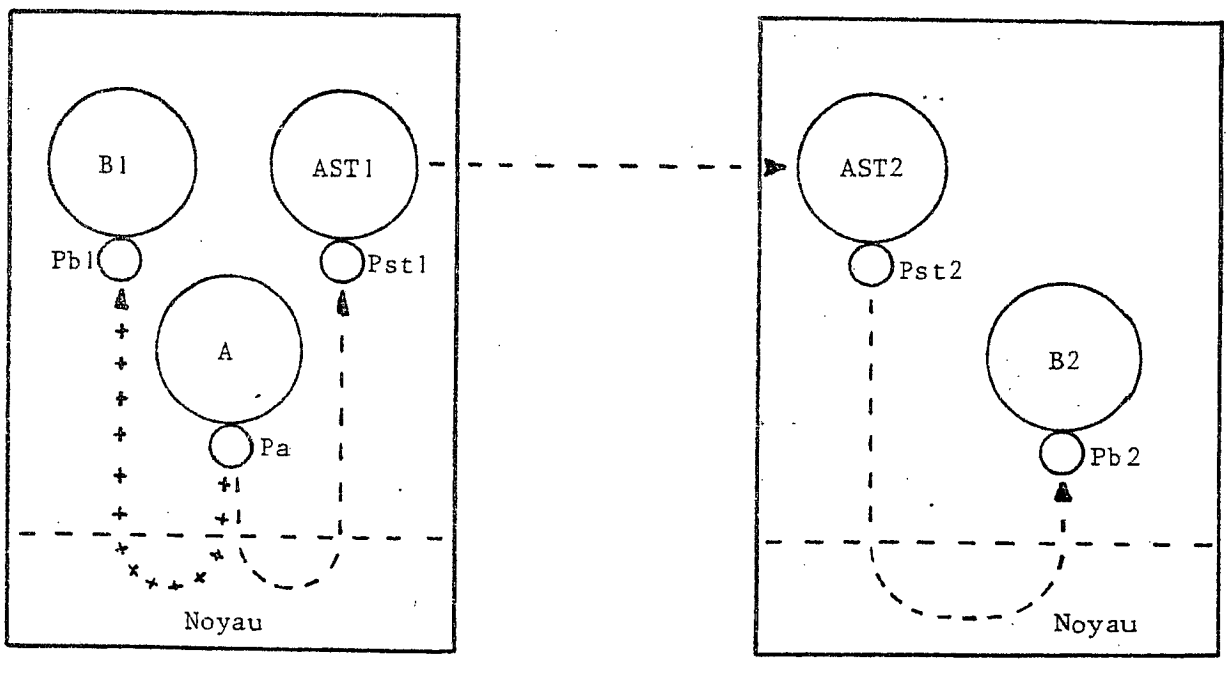

figure 3.5 : Communications locales et distantes

Routage

Les acteurs Station de Transport (AST) mettent en oeuvre, pour chercher le site de résidence d'une porte, un algorithme de routage (décrit ci-dessous) qui a été conçu de façon à être simple, même s'il n'est pas toujours optimal. L'un des objectif de cet algorithme est de permettre aux AST de suivre les éventuelles migrations des portes sans avoir à gérer de grandes tables.

Notons bien que ce paragraphe décrit l'algorithme de routage et de transport entre les AST et non pas le protocole de transport entre les acteurs d'application!

Il y a un AST sur chaque site. Chaque AST gère trois tables:

- les noms uniques de toutes les portes locales ouvertes (table A).
- les noms uniques de portes distantes avec leur site de résidence (table B). Quand un AST commence à travailler, cette table est vide; la mise à jour de cette table est décrite ci-dessous.
- les adresses réseau des autres AST (table C) ou une adresse de diffusion.

L'algorithme de routage est le suivant:

(1) Quand un AST reçoit "Pa, Pb, [Texte du message]" de son noyau local, deux cas peuvent se produire:

a/ Pb se trouve dans la table B (cf figure 3.6): l'AST envoie le message sur le réseau vers le site de résidence de Pb et il attend un acquittement.

- Si l'AST reçoit l'acquittement, la communication est terminée.
- Si l'AST ne reçoit pas d'acquittement, il enlève Pb de la table B et poursuit l'algorithme en b/ (ci-dessous).

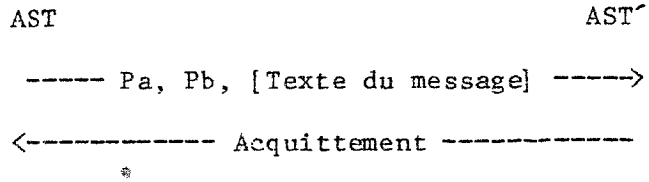

figure 3.6 : Le protocole de transport distant lorsque Pb est dans la table B

b/ Pb ne se trouve pas dans la table B (cf figure 3.7): l'AST interroge les autres AST (en utilisant sa table C) pour chercher sur quel site se trouve Pb.

- Si un AST' répond positivement, l'AST envoie le message sur le réseau vers AST' et il met à jour sa table B.
- Si aucun AST' ne répond positivement, la porte Pb est considérée comme inaccessible et le message est perdu.

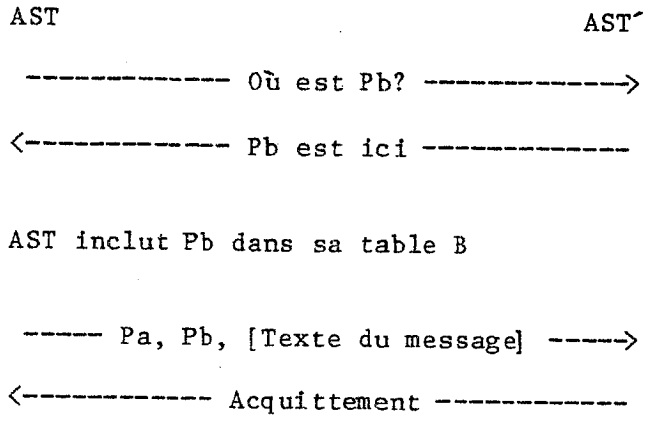

figure 3.7 : Le protocole complet de transport distant

(2) Lorsqu'un AST' reçoit "Pa, Pb, [Texte du message]" du réseau (c.a.d. d'un AST distant), deux cas peuvent se produire:

a/ Pb se trouve dans la table A, c.a.d. Pb est une porte locale ouverte (cf figure 3.6): l'AST' envoie le message localement (vers Pb) et renvoie un acquittement sur le réseau vers l'AST distant.

b/ Pb ne se trouve pas dans la table A (cf figure 3.8): l'AST' informe l'AST distant que Pb n'est plus sur le site local (avec un acquittement négatif); l'AST enlève Pb de sa table B et poursuit l'algorithme comme dans le cas 1-b/ ci-dessus.

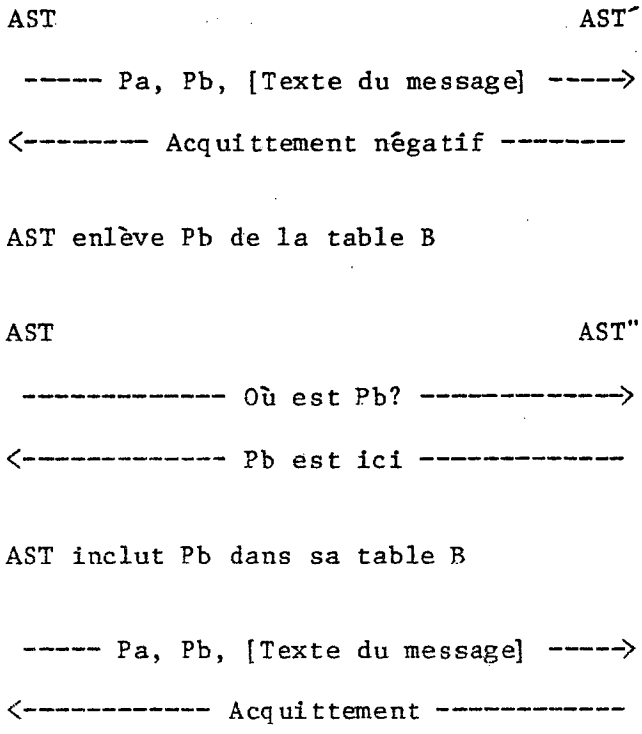

figure 3.8 : Le protocole de transport distant lorsque Pb a migré

Note 1: les messages envoyés sur le réseau sont numérotés séquentiellement: ainsi, l'AST récepteur peut détecter les duplications éventuelles.

Note 2: un AST utilise des temporisations pour détecter la non-réception d'un acquittement, qui signifie la perte d'un message, d'un acquittement ou la panne d'un AST distant.

Note 3: la table B (c.a.d. la table des portes distantes) a une taille limitée. Quand elle est pleine, le nom de porte le moins fréquemment utilisé est enlevé de cette table.

Note 4: quand un nouveau site est connecté au réseau, chaque AST met à jour sa table C (c.a.d. la table des adresses réseau des autres AST); le nouveau site devient accessible ainsi que toutes ses portes.

4/ Discussion des choix de CHORUS

Ce paragraphe discute et justifie les choix effectués lors de la conception de l'architecture CHORUS; une comparaison entre CHORUS et d'autres systèmes d'exploitation répartis peut être trouvée dans [Guillemont 84].

4.1/ La synchronisation

Dans un système réparti, la synchronisation ne peut pas être assurée par un mécanisme central unique comme dans les systèmes centralisés. Les objectifs d'efficacité et de robustesse imposent de disposer d'un mécanisme réparti basé sur l'échange de messages: une exécution répartie est naturellement guidée par les messages et les messages constituent l'outil élémentaire de synchronisation.

CHORUS propose une vision intégrée de la synchronisation et des communications qui sont clairement distinguées des traitements: l'envoi d'un message est asynchrone (c.a.d. non bloquant), la réception d'un message déclenche une étape de traitement. Ce mécanisme est compatible avec la répartition (puisque la communication est indépendante de la localisation) et il introduit le plus faible couplage entre les acteurs: l'étape de traitement qui a produit le message doit être terminée avant que le message soit émis et qu'il déclenche une nouvelle étape de traitement.

Ce mécanisme favorise la programmation "asynchrone", bien adaptée à la répartition mais qui n'est pas offerte par les langages modernes de haut niveau qui reposent au contraire sur l'"appel de procédure" (synchrone). Cependant, ce dernier schéma de communication (l'appel de procédure) peut facilement être construit au-dessus des mécanismes CHORUS, comme cela est expliqué ci-dessous.

L'appel de procédure externe

Un acteur A demande l'exécution d'une procédure (c.a.d. une étape de traitement) dans un acteur B et il attend que B ait terminé pour reprendre sa propre exécution. Plus précisément, A et B utilisent le protocole "Demande-Réponse" (cf figure 4.1):

- l'acteur A envoie "Pa, Pb, [Demande]" et s'arrête,
- l'acteur B traite la [Demande] et envoie "Pb, Pa, [Réponse]",
- l'acteur A reçoit [Réponse] et il reprend son exécution.

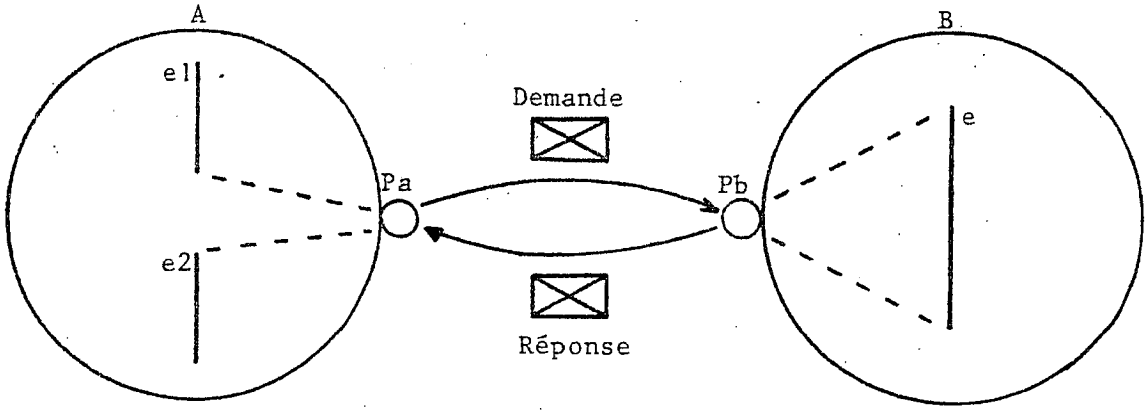

figure 4.1 : L'appel de procédure externe

Cette synchronisation est programmée de la façon suivante:

```
acteur A                                    acteur B

                                            .....
                                            RETOURNER (Pb, Sélection, [(Pb, Pa)];
                                                       Pb, Aiguillage, [e] ;
                                                       .....);

POINT-D'ENTREE e1;
 .....
RETOURNER (Pa, Pb, [Demande];
           Pa, Sélection, [(Pb, Pa)];
           Pa, Aiguillage, [e2] );

                --- Pa, Pb, [Demande] --->

                                            POINT-D'ENTREE e;
                                            .....
                                            { traitement de [Demande] }
                                            { constitution de [Réponse] }
                                            .....
                                            RETOURNER (Pb, Pa, [Réponse] ;
                                                       .....);

                <--- Pb, Pa, [Réponse] ---

POINT D'ENTREE e2;
 .....
```

figure 4.2 : Programmation d'un appel de procédure externe

A la fin de l'étape de traitement e1, l'acteur A envoie la [Demande] à Pb; la condition de sélection donnée par A signifie que seul un message envoyé de Pb à Pa (c.a.d. [Réponse]) peut être sélectionné; la condition d'aiguillage

signifie que le message reçu sur Pa doit déclencher l'étape de traitement e2.

Ainsi, ceci réalise l'"appel de procédure externe": A exécute e1, demande l'exécution d'une étape de traitement dans B (ici "e") et reprend son exécution en e2 lorsqu'il reçoit la réponse de B.

Note: cette façon de programmer peut être dangereuse car l'acteur A sera indéfiniment bloqué si B tombe en panne avant d'avoir envoyé son message de réponse. Afin d'éviter cette situation, la programmation correcte de A est la suivante:

```
RETOURNER (Pa, Pb, [Demande] ;
           Pa, Sélection, [(Pb, Pa)] ;
           Pa, Aiguillage, [e2] ;
           Pa, Temporisation, [Pb, Pa, délai] );
POINT-D'ENTREE e2;
```

Si B tombe en panne, la temporisation sur Pa relancera l'exécution de A en e2.

. Cette séquence particulière de code qui permet d'enchainer séquentiellement deux étapes de traitement dans un acteur a été nommée APPEL_EXTERNE. L'acteur A peut ainsi être programmé de la façon suivante:

```
POINT-D'ENTREE e1;
.....
.....  (étape de traitement e1)
.....
APPEL_EXTERNE (Pa, Pb, [Demande], délai);
.....
.....  (étape de traitement e2)
.....
RETOURNER (.....);
```

Bien sûr, ceci est particulièrement commode pour programmer des acteurs en Pascal, par exemple: l'expression de l'appel d'une procédure externe est analogue à l'expression de l'appel d'une procédure interne.

4.2/ Désignation

Le mécanisme de désignation utilisé dans CHORUS repose sur l'usage de noms globaux uniques (cf paragraphe 3.5). Ceci offre plusieurs avantages:

- ce mécanisme est fiable en ce sens qu'il ne peut y avoir aucune confusion dans la relation nom global/entité.
- il n'est pas nécessaire d'établir des correspondances de noms entre les acteurs ou entre les sites.

Ici encore, le choix de CHORUS est le mécanisme le plus élémentaire et le plus fiable: les autres mécanismes, plus rapides ou plus spécifiques, peuvent être construits au-dessus de celui-ci (mais non l'inverse); et si ces autres mécanismes sont défaillants, le système est toujours capable de revenir à la désignation de base pour repartir.

Des exemples de deux autres mécanismes de désignation sont donnés ci-dessous:

Accélérateurs

Ce mécanisme est conçu pour être plus rapide et moins coûteux que l'utilisation des noms globaux.

Un acteur peut définir des noms locaux qui désignent

- ses propres portes.
- des liaisons entre ses portes et des portes distantes (c.a.d. des couples de portes).

Les noms locaux sont interprétés uniquement par l'acteur et le noyau local. Ils permettent d'accélérer la recherche d'une porte dans les tables pour le routage, le contrôle et la mise en file d'attente des messages. Cependant, les

messages transportés d'une porte vers une autre porte sont toujours accompagnés des noms globaux uniques de ces portes.

Noms fonctionnels et noms de groupe

Un acteur système, l'acteur Gestionnaire des Noms, offre la possibilité de définir deux sortes de noms [Senay 83] :

- LES NOMS FONCTIONNELS (1 parmi n). Plusieurs portes d'une application peuvent offrir des services équivalents; quand un acteur demande ce service, n'importe laquelle de ces portes peut traiter le message de requête. Ces portes ont, en plus de leur nom global unique, un nom fonctionnel commun Pf: selon le schéma donné au paragraphe 3.5, le "type" de Pf est "nom fonctionnel". Quand un acteur demande le service, il envoie un message vers Pf; l'AST local (acteur Station de Transport) reçoit ce message, localise une portes de ce nom (éventuellement locale) et envoie le message sur cette porte. La localisation de la porte se fait avec le protocole présenté au paragraphe 3.6: l'AST diffuse le message [Où est Pf?] et il ne retient que la première réponse (qu'il ne mémorise pas dans sa table des portes distantes).

Ce même mécanisme est également utilisé lorsqu'il y a une porte sur chaque site pour le même service et qu'un acteur doit toujours s'adresser à la porte locale (c'est le cas de la plupart des services système): toutes ces portes ont le même nom fonctionnel Pf et l'envoi d'un message sur Pf atteint toujours la porte Pf qui est sur le même site que l'émetteur. Dans ce cas, ce mécanisme simplifie l'"édition de lien" dynamique entre un acteur et les services système auxquels il accède.

- LES NOMS DE GROUPE (n parmi n). Un acteur peut demander la diffusion d'un message sur un ensemble de portes. Cet ensemble de portes a un nom de groupe Pg: en fait, aucune porte dans le système ne porte ce nom Pg; simplement, Pg représente un groupe de portes et Pg a la structure d'un nom de porte. Selon le schéma donné au paragraphe 3.5, le "type" de Pg est "nom de groupe". Chaque AGN

(acteur Gestionnaire de Noms) gère une liste partielle de portes appartenant au groupe Pg. Quand un acteur envoie un message sur Pg, l'AST local reçoit le message; il demande à l'AGN local sa liste des portes appartenant à Pg et il envoie une copie du message à toutes les portes de cette liste; simultanément, il diffuse à tous les autres AST une copie du message et chaque AST envoie une copie du message à toutes les portes de la liste gérée par l'AGN local. Ce mécanisme est récurrent: une liste des portes appartenant à un groupe peut contenir le nom d'un autre groupe.

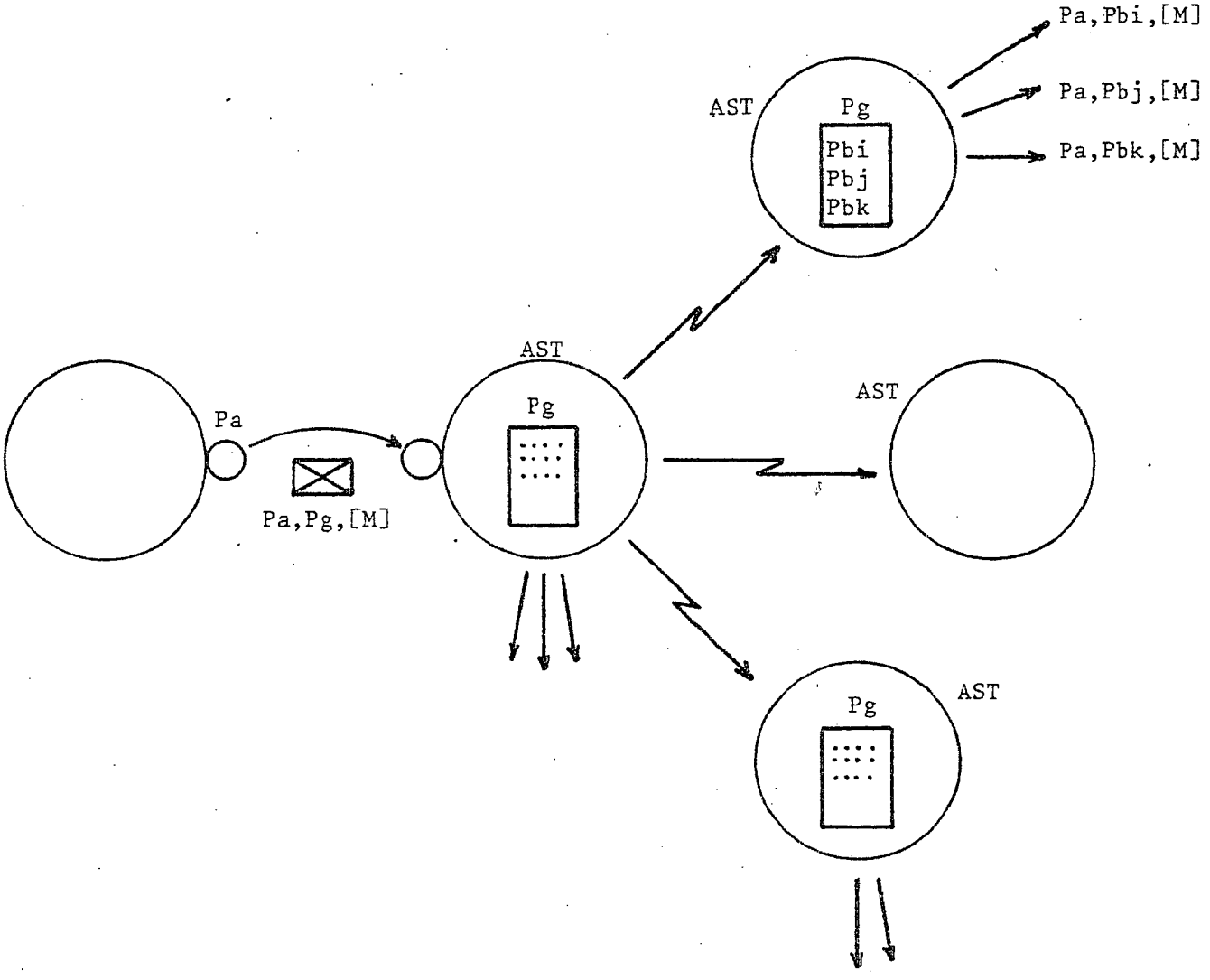

figure 4.3 : Les noms de groupe

4.3/ Protection

La répartition impose des limites sévères aux contrôles qu'il est possible de réaliser à la compilation et elle rend nécessaire d'autres contrôles à l'exécution. L'absence d'un contrôle central tend à rendre les systèmes répartis plus complexes et impose des mécanismes de protection plus robustes ou du cryptage pour réduire la propagation des erreurs et des pannes. D'autre part, le mécanisme traditionnel de capacités nécessite un matériel spécialisé qui n'est pas toujours disponible, même sur les machines modernes. Ceci suggère qu'un mécanisme plus simple peut être plus adéquat pour des systèmes répartis qui ont, en outre, la possibilité d'isoler physiquement un site pour le protéger.

Le mécanisme de protection adopté dans CHORUS est construit en deux niveaux (cf paragraphe 2.5):

- le système contrôle que l'architecture est respectée (c.a.d. contrôle principalement les relations acteur/porte).
- des procédures de contrôle décrivent les autres contrôles pour chaque application.

Le premier niveau de protection permet, par exemple, à un acteur de baser sa protection sur une liste de portes authentifiées qui sont autorisées à communiquer avec lui; l'acteur associé à une porte ne peut pas changer sauf si la porte est fermée puis re-ouverte; mais ce dernier service système est contrôlé par le second niveau de protection.

Dans les communications, les procédures de contrôle d'émission et de réception associées aux portes émettrice et réceptrice permettent de contrôler des mots de passe dans les messages, de crypter et décrypter les messages, de contrôler des types (s'ils sont apparents dans les messages), etc...

Les procédures de contrôle sont écrites par les utilisateurs: la nature du contrôle peut être adaptée à chaque application. Cela permet par exemple que les procédures de contrôle et les modèles d'acteur soient écrits par des équipes

différentes. Cela permet également de supprimer certains contrôles lors du passage d'une phase de mise au point à une phase opérationnelle sans modifier le code des acteurs. CHORUS n'impose que la liste des procédures de contrôle et le moment où elles sont exécutées.

4.4/ Reconfiguration

L'exécution d'un acteur est une suite d'étapes de traitement. Pendant l'exécution d'une étape de traitement, un acteur ne peut ni envoyer ni recevoir de messages: un message déclenche l'étape de traitement et les messages envoyés ne le sont qu'à la fin de l'étape de traitement. Par conséquent, lorsqu'un acteur reçoit un message, il est certain que l'étape de traitement qui a constitué ce message s'est correctement terminée; réciproquement, si une étape de traitement ne se termine pas correctement, aucun message n'est envoyé.

Un mécanisme de résistance aux pannes, basé sur cette simple remarque, a été développé ([Banino 82], [Fabre 82]): l'idée première de ce mécanisme est qu'un service fiable peut être rendu par deux acteurs identiques répartis sur deux sites (le Maître et l'Esclave); le Maître reçoit les messages de demande et l'Esclave reçoit une copie de ces messages renvoyées par le Maître; ainsi, les deux acteurs recoivent les mêmes messages de demande, dans le même ordre et tous deux exécutent les mêmes étapes de traitement, en parallèle; mais un seul acteur (le Maître) envoie les messages de réponse. Les deux acteurs restent donc toujours dans le même état et le client (du service fiable) ne reçoit qu'une seule réponse. Si le Maître tombe en panne, l'Esclave devient Maître exactement au début de l'étape de traitement au cours de laquelle le Maître est tombé en panne; grâce à l'"atomicité" d'une étape de traitement, le Maître n'a envoyé aucun des messages qu'il avait préparés au cours de sa dernière étape de traitement; comme l'Esclave devient le nouveau Maître exactement au début de cette étape de traitement, il n'y a ni perte ni duplication de message pour le client: le nouveau Maître a reçu une copie du message de demande - pendant qu'il était Esclave - et il enverra sa réponse puisqu'il est maintenant le

Maître.

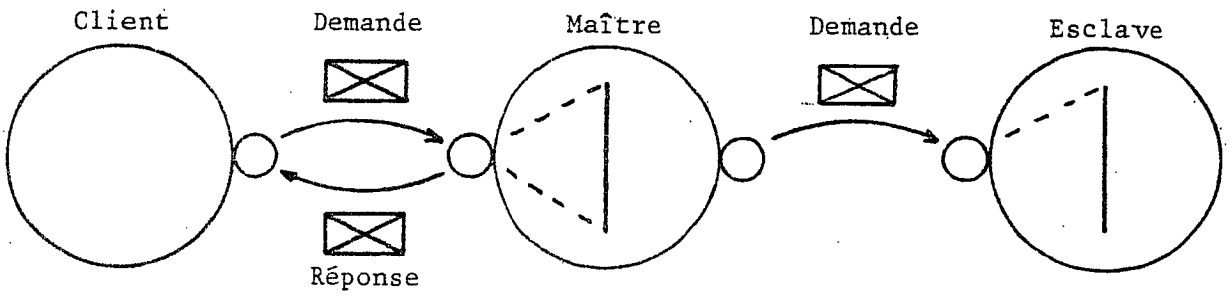

figure 4.4 : Les acteurs couplés répartis

4.5/ Hétérogénéité

L'architecture CHORUS est indépendante du matériel sur lequel elle est implantée ainsi que du support de communication qui relie les sites. L'architecture CHORUS repose sur un noyau résident sur chaque site qui offre une interface standard. Ainsi, CHORUS peut être implanté sur des machines différentes: chaque machine a son noyau spécifique et quelques acteurs système doivent être adaptés (par exemple l'acteur Gestionnaire de Mémoire).

Quelques remarques à propos de l'hétérogénéité:

(1) La communication entre des machines hétérogènes doit tenir compte du codage interne des messages: ordre des bits de poids fort et faible d'un octet, ordre des octets dans un mot, nombre d'octets pour un entier, etc...

(2) Un ensemble de machines CHORUS peut très bien être connecté à une machine qui ne supporte pas l'architecture CHORUS, à condition que cette dernière soit vue comme un ensemble de portes qui envoient et recoivent des messages. Dans ce cas, les procédures de contrôle d'émission et de réception seront particulièrement utiles pour la protection. Par exemple, dans une machine multi-processeur, certains processeurs peuvent respecter l'architecture CHORUS tandis que d'autres (les processeurs d'E/S par exemple) ne le font pas: les processeurs du second groupe peuvent néanmoins envoyer et recevoir des

messages sur des portes.

(3) Si l'architecture CHORUS est indépendante du matériel, les performances ne le sont pas: il est évident, par exemple, que la diffusion sur un réseau du type Ethernet est plus performante que la diffusion sur un réseau maillé à longue distance! En d'autres termes, la diffusion peut être disponible quel que soit le support de communication, mais les utilisateurs doivent savoir que ses performances dépendent de l'implantation.

4.6/ La production de logiciel

La figure 2.6 présente une description des étapes de traitement directement issue du fonctionnement d'un acteur. Cette représentation est trop rigide pour constituer un outil de programmation des modèles d'acteurs.

Les implantations actuelles de CHORUS sont écrites en Pascal. Afin d'écrire des modèles d'acteur, CHORUS a développé un ensemble d'outils qui facilitent la programmation. Ces outils sont résumés ci-dessous:

1/ un modèle d'acteur est un programme Pascal où chaque étape de traitement est représentée par une procédure. Les compilateurs standards sont utilisés pour compiler les modèles d'acteur.

2/ un modèle d'acteur contient trois variables standards, par exemple "Porte_Emettrice", "Porte_Réceptric" et "Message", qui contiennent, au lancement de chaque étape de traitement, la description du message reçu; les anciennes valeurs de ces trois variables sont écrasées au début de chaque nouvelle étape de traitement.

3/ un modèle d'acteur définit (implicitement) une liste "Liste" où tous les messages qui doivent être émis à la fin de l'étape de traitement sont empilés; des procédures d'interface empilent les messages dans cette liste et même construisent aussi les messages. Voici quelques procédures d'interface:

PREPARER (P, Q, M)

empile "P, Q, [M]"

SELECTIONNER (Q, P)

construit et empile "P, Selection, [(Q, P)]"

AIGUILLER (P, E)

construit et empile "P, Aiguillage, [E]"

TEMPORISER (P', Q, P, Délai)

construit et empile "P', Temporisation, [Q, P, Délai]"

CREER_PORTE (P, Modèle, Nom, Paramètres d'initialisation)

construit et empile "P, Creation_de_Porte, [Modèle, Nom, Paramètres d'initialisation]"

etc...

4/ RETOURNER est une procédure d'interface sans paramètres. Quand un acteur appelle RETOURNER, le contrôle est rendu au noyau et la "Liste" des messages lui est transmise.

Avec ces règles, une étape de traitement et un modèle d'acteur sont programmés de la façon suivante:

```
PROCEDURE entrée;
.....
SI ... ALORS TEMPORISER (Q1, P1, Délai)
       SINON PREPARER (P2, Q2, Mess);
POUR i := 1 A n FAIRE PREPARER (Pi, Qi, Mi);
.....
RETOURNER;
END;
```

figure 4.5 : Une étape de traitement

```
PROGRAMME modèle_d'acteur;

PROCEDURE entrée_1;

.....

END;

PROCEDURE entrée_2;

.....

END;

etc...
```

figure 4.6 : Un modèle d'acteur

Les procédures d'interface du premier ensemble présenté ci-dessus construisent et empilent chacune un message.

Il existe un second ensemble de procédures d'interface qui réalisent des fonctions plus complexes. En particulier, l'"appel de procédure externe" (cf paragraphe 4.1) est disponible sous forme d'une procédure APPEL_EXTERNE; CHORUS offre un second ensemble de procédures d'interface (construites sur cette procédure) qui demandent l'exécution synchrone des différents services système. Par exemple, la procédure OUVRIR_PORTE_SYNCHRONE demande, de façon synchrone, l'ouverture d'une porte; cette procédure est programmée de la façon suivante:

```
OUVRIR_PORTE_SYNCHRONE (Porte, Nom, Priorité, VAR Diagnostic);

BEGIN
Sauvegarder "Porte_Emettrice", "Porte_Réceptrice", "Message";
Sauvegarder "Liste";
APPEL_EXTERNE (Porte, Ouvrir_Porte, [Nom, Priorité], Délai);
(* Après l'exécution de cette procédure, le message reçu
   se trouve dans les variables "Porte_Emettrice", etc...  *)
SI Porte_Emettrice = Temporisation
     ALORS Diagnostic := "Temporisation échue"
     SINON Extraire Diagnostic de Message;
Restaurer "Liste";
Restaurer "Porte_Emettrice", "Porte_Réceptrice", "Message";
END;
```

Les procédures d'interface du premier ensemble sont les "procédures d'interface asynchrones", les procédures d'interface du second ensemble sont les "procédures d'interface synchrones". Ce second ensemble est tout à fait adapté à la programmation dans des langages basés sur l'appel de procédure (comme Pascal): l'appel d'un service système est réalisé par l'appel d'une procédure d'interface synchrone.

Un ensemble complet de procédures d'interface est donné en annexe.

5/ Illustration de l'architecture CHORUS

Ce paragraphe est bâti autour d'un exemple de messagerie très simple. Cette exemple ne constitue pas notre approche d'une messagerie répartie; il ne s'agit que d'un exemple destiné à illustrer l'approche CHORUS; à cette fin, cet exemple est volontairement simple, voire naif.

5.1/ Structure de la mini-messagerie

La mini-messagerie est constituée d'acteurs de deux modèles:

- les serveurs de messagerie SM qui sont des acteurs permanents, à raison d'un par site.
- les utilisateurs de la messagerie UM qui sont des acteurs temporaires, à raison d'un par utilisateur; ils peuvent apparaitre sur n'importe quel site.

Chaque utilisateur (humain) a une boîte à lettres.

Chaque serveur de messagerie

- gère un ensemble de boîtes à lettres et les messages dans chaque boîte,
- surveille un autre serveur de messagerie SM'; si SM' tombe en panne, les messages qui sont dans les boîtes à lettres gérées par SM' sont temporairement inaccessibles (ils peuvent avoir été sauvegardés sur une mémoire stable), mais SM crée des "copies" des boîtes à lettres gérées par SM' de telle sorte que ces copies puissent recevoir les nouveaux messages qui leur sont envoyés; ainsi, ces boîtes à lettres sont à nouveau accessibles, à la restriction près que les messages reçus avant la panne ne sont plus accessibles.

Un utilisateur de la messagerie UM est créé chaque fois qu'un utilisateur du système veut utiliser la mini-messagerie. Un UM peut

- envoyer un message dans une boîte à lettres,
- lire les messages dans sa boîte à lettres.

Chaque boîte à lettres est représentée par une porte: la boîte à lettres de l'utilisateur U est représentée par la porte BalU (Boîte à lettres de l'utilisateur U). La mini-messagerie est vue par les utilisateurs comme un ensemble de portes BalU; chaque utilisateur connaît le nom de sa boîte à lettres et peut le communiquer à ses amis. Les portes BalU sont des portes des serveurs de messagerie; le SM qui a ouvert la porte BalU gère la boîte à lettres de l'utilisateur U.

Les serveurs de messagerie ont aussi des portes privées PpS (Porte privée du serveur S) dont les noms ne sont pas connus des utilisateurs. Le seraient-ils même que les procédures de contrôle de réception des PpS rejetteraient les messages incongrus (cf paragraphe 5.4).

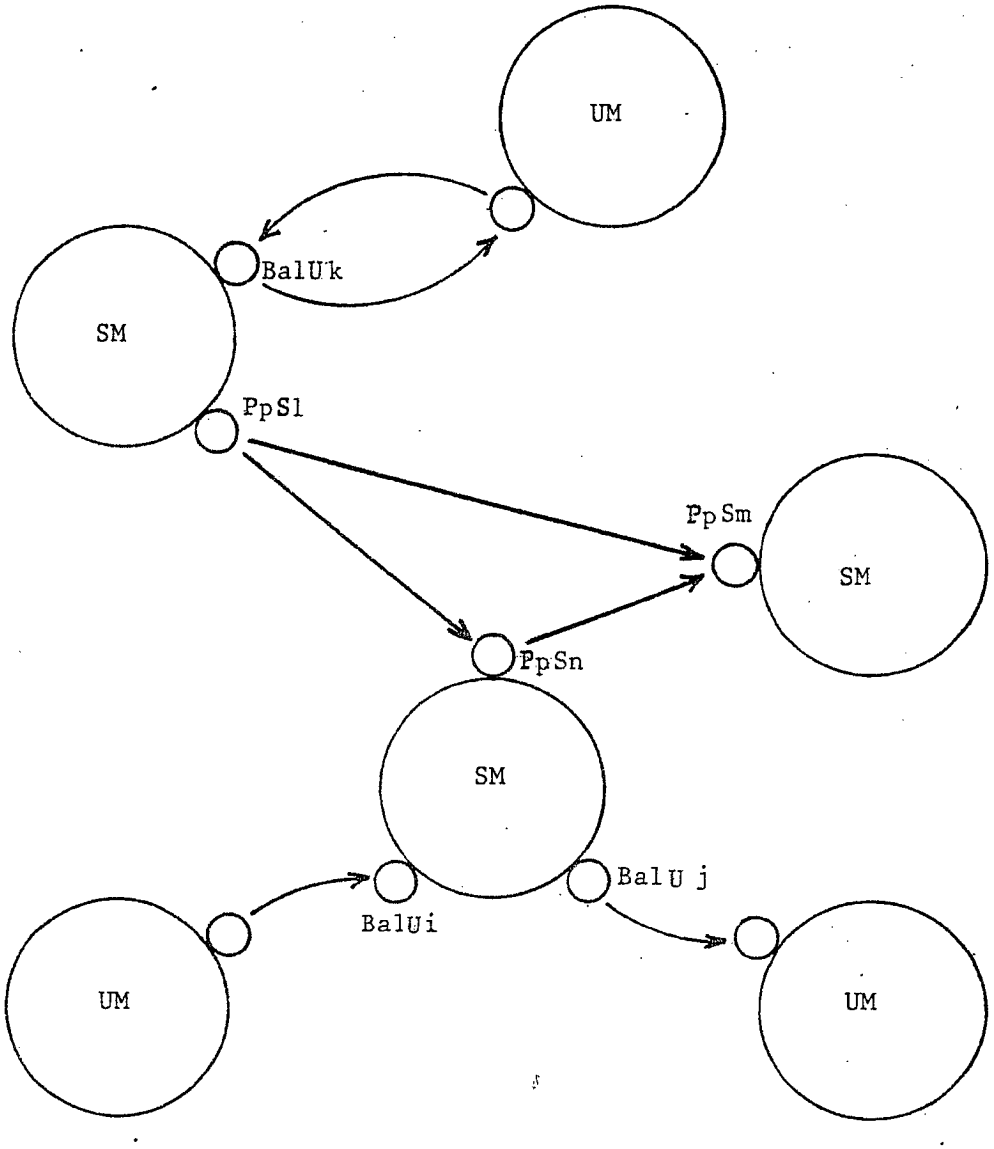

figure 5.1 : La mini-messagerie

Usage des boîtes à lettres

(*) Un acteur utilisateur de la messagerie UM demande la création d'une boîte à lettres pour un utilisateur humain en envoyant

Porte, Création_Bal, [Création, Utilisateur, Mot de passe MdpU]

Création_Bal est un nom fonctionnel de porte des SM. La mini-messagerie crée un un nouveau nom unique BalU pour la boîte à lettres et l'acteur SM répond

Création_Bal, Porte, [BalU]

(*) Un acteur UM envoie un message dans la boîte à lettres BalU en envoyant

Porte, BalU, [Envoi, Message]

(*) Un acteur UM lit une boîte à lettres BalU en envoyant

Porte, BalU, [Lecture, Mot de passe MdpU]

l'acteur SM qui gère BalU répond

BalU, Porte, [Lecture, Message]

(*) Un acteur UM demande la destruction d'une boîte à lettres en envoyant

Porte, BalU, [Destruction, Mot de passe MdpU]

5.2/ Synchronisation

Chaque boîte à lettres est gérée par un serveur de messagerie SM: l'absence de parallélisme à l'intérieur d'un acteur garantit une exclusion mutuelle des accés à la boîte à lettres; deux utlisateurs de la messagerie peuvent envoyer simultanément deux messages dans BalU, leurs requêtes seront traitées séquentiellement; de même, un utilisateur de la messagerie peut lire sa boîte à lettres pendant qu'un autre utilisateur lui envoie des messages, leurs requêtes seront traitées séquentiellement.

5.3/ Désignation

Chaque boîte à lettres reçoit un nom global unique BalU. Ces noms sont indépendants des sites; les portes sont nomades.

L'accès à une boîte à lettres ne fait aucune hypothèse sur la localisation du serveur de messagerie qui la gère. De même, une boîte à lettres peut migrer d'un serveur de messagerie à un autre (cf paragraphe 5.5) sans modifier l'accés à cette boîte à lettres.

5.4/ La protection

La protection concerne les boîte à lettres et les serveurs de messagerie.

Protection des boîtes à lettres

Chaque boîte à lettres est créée avec un mot de passe MdpU (cf paragraphe 5.1). La porte BalU reçoit ce mot de passe MdpU à sa création (cf paragraphe 2.3.3). Tous les messages d'accès à la boîte à lettres sont reçus sur BalU. La procédure de contrôle de réception de BalU protège la boîte à lettres:

- les messages qui ne correspondent pas à une opération connue (envoi, lecture, destruction, etc...) sont rejetés.
- les messages (excepté pour l'envoi) qui ne contiennent pas MdpU sont rejetés.

Protection des serveurs de messagerie

Un mécanisme analogue protège les serveurs de messagerie.

Les serveurs de messagerie communiquent à travers des portes privées PpS; à chaque PpS est associé un mot de passe MdpS.

Quand un serveur de messagerie est créé (par un autre serveur de messagerie) il reçoit dans son message initial la liste des autres couples (PpS', MdpS'); le nouveau serveur de messagerie créé et ouvre sa porte PpS et il envoie à chaque PpS'

PpS, PpS', [mon MdpS, votre MdpS']

Ce message authentifie PpS pour les autres serveurs de messagerie et il leur transmet le MdpS associé. Ainsi, par récurrence, les serveurs de messagerie constituent un groupe de portes authentifiées.

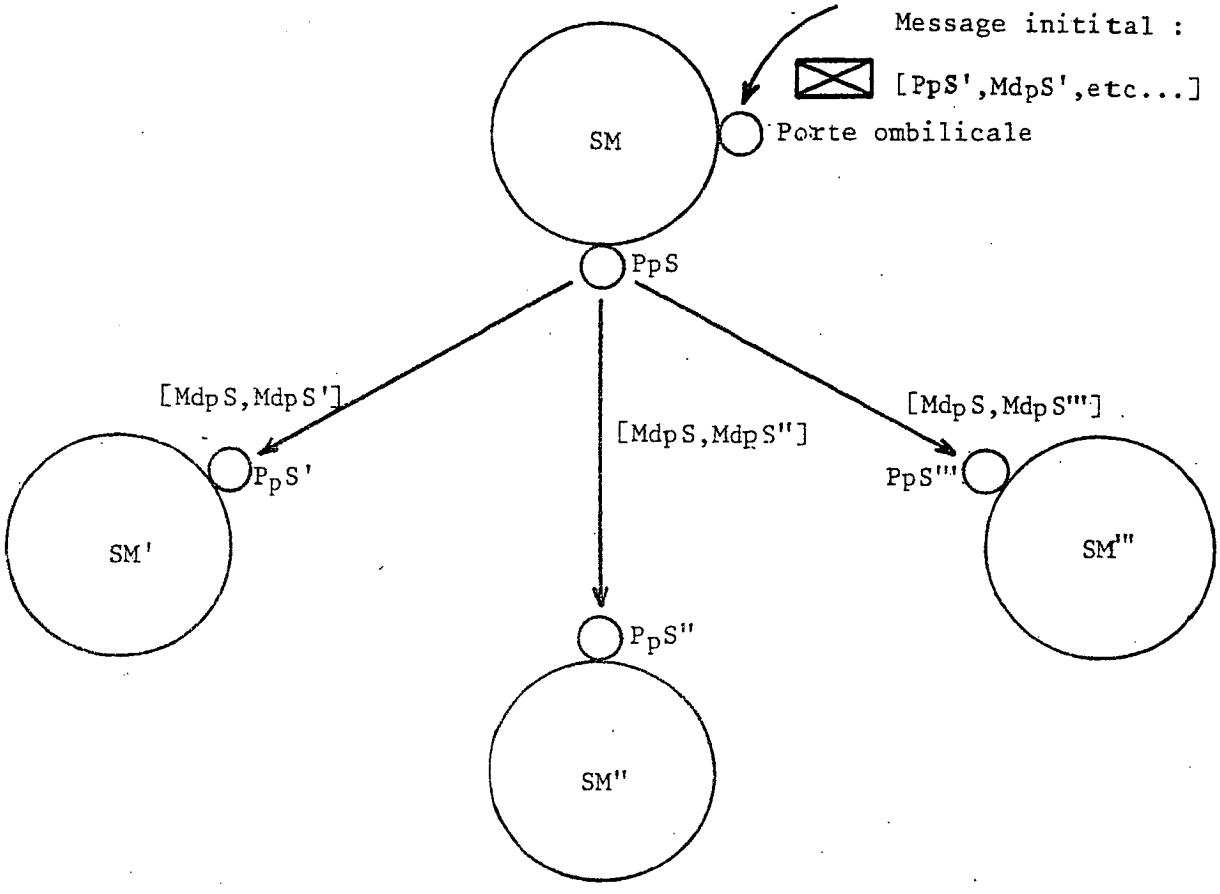

figure 5.2 : Authentification d'une PpS

Chaque nouveau message envoyé sur PpS devra

- être envoyé par l'une des PpS'
- contenir MdpS.

Ce dernier contrôle peut être effectué par la procédure de contrôle de réception de PpS.

L'ensemble des serveurs de messagerie constitue une activité. La protection et la reconfiguration (cf paragraphe 5.5) sont définis sur la base de cette activité.

5.5/ Reconfiguration

La reconfiguration est basée sur

- une surveilance mutuelle des serveurs de messagerie,
- la migration des portes BalU.

Soit SM1 un serveur de messagerie qui surveille SM2; SM2 gère les boîtes à lettres BalU1, ..., BalUp. La reconfiguration, en cas de panne, peut se résumer de la façon suivante:

1/ SM2 envoie périodiquement à SM1

PpS2, PpS1, [liste des BalUi, MdpS1]

2/ SM1 réarme une temporisation à la réception de chaque message de PpS2 sur PpS1

P, Temporisation, [PpS2, PpS1, Délai]

3/ Si SM2 tombe en panne, la temporisation arrive à échéance et SM1 détecte la panne de SM2. Comme SM2 est tombé en panne, toutes ses portes, et en particulier les BalUi, ont été automatiquement fermées par le système. SM1 ouvre ces portes

PpS1, Ouverture_de_Porte, [BalUi, Priorité]

Pour les utilisateurs de la messagerie, les boîtes à lettres restent disponibles et leur accès est inchangé. La seule différence est que les messages reçus par SM2 sur BalUi sont (temporairement) inaccessibles.

4/ Quand SM2 réapparaît, SM2 envoie un message à SM1; SM1 ferme les portes BalUi et il envoie à SM2

PpS1, PpS2, [liste des BalUi, liste des messages reçus sur chaque BalUi, MdpS2]

SM2 réouvre les portes BalUi et le service est à nouveau entièrement disponible.

Avec des mécanismes analogues, le service de messagerie peut être étendu en ajoutant de nouvelles boîtes à lettres ou en créant de nouveaux serveurs de messagerie.

5.6/ Programmation des protocoles

Exemple d'un contrôle de flux sans anticipation entre un serveur de messagerie et un utilisateur de la messagerie:

Un utilisateur de la messagerie veut lire sur sa porte "Porte" tous les messages de la boîte à lettres BalU. Le serveur de messagerie et l'utilisateur de la messagerie travaillent en co-routines:

- le serveur de messagerie envoie un message et attend son acquittement pour envoyer le message suivant,
- l'utilisateur de la messagerie envoie un acquittement et attend le message suivant.

Serveur de messagerie:

```
POINT-D'ENTREE e;
SI fin_de_boîte_à_lettres ALORS M := Fin_de_bal
SINON M := Message_suivant;
RETOURNER (BalU, Porte, [M] ;
           BalU, Sélection, [(Porte, BalU)] );
```

Utilisateur de la messagerie:

```
POINT-D'ENTREE e;
SI M = Fin_de_bal ALORS ...;
traitement de M;
RETOURNER (Porte, BalU, [Acquittement, MdpU] ;
           Porte, Sélection, [(BalU, Porte)] );
```

Afin de traiter les éventuelles pertes de messages, le serveur de messagerie et l'utilisateur de la messagerie peuvent être programmés de la façon suivante: l'algorithme est le même que ci-dessus, mais les acteurs arment des temporisations sur leurs portes et testent si la temporisation est arrivée à échéance ou non lorsqu'ils s'exécutent. (Si Porte_émettrice = Temporisation, le message traité est une indication de temporisation échue; si Porte_émettrice ◇ Temporisation, le message traité est un message normal.)

Serveur de messagerie:

```
POINT-D'ENTREE e;
SI Porte_émettrice = Temporisation
     ALORS M := dernier_message_envoyé
SINON SI fin_de_boîte_à_lettres ALORS M := Fin_de_bal
     SINON M := Message_suivant;
     dernier_message_envoyé := M;
RETOURNER (BalU, Porte, [M] ;
           BalU, Sélection, [(Porte, BalU)] ;
           BalU, Temporisation, [Porte, BalU, Délai] );
```

Utilisateur de la messagerie:

```
POINT-D'ENTREE e;
SI Porte_émettrice ◇ Temporisation ALORS
     SI M = Fin_de_bal ALORS ...;
     SINON traitement de M;
RETOURNER (Porte, BalU, [Acquittement, MdpU] ;
           Porte, Sélection, [(BalU, Porte)] ;
           Porte, Temporisation, [BalU, Porte, Délai] );
```

Remarquons que même si l'étape de traitement de l'utilisateur de la messagerie est déclenchée par l'échéance d'une temporisation, il envoie un

acquittement: ceci permet de traiter le cas de la perte d'un acquittement; ceci implique aussi que l'utilisateur de la messagerie doit être capable de détecter la réception de deux messages consécutifs identiques (il suffit de numéroter séquentiellement les messages).

6/ Implantations de CHORUS

CHORUS a été implanté sur un ensemble d'Intel 8086 reliés par le réseau à diffusion Danube [Naffah 80].

Une seconde implantation est en cours sur un ensemble de SM90 [Finger 82], une machine multi-processeurs à base de Motorola 68000.

Les deux implantations sont écrites en Pascal.

6.1/ Implantation sur Intel 8086

Cette implantation utilise le système de développement Pascal-UCSD [Softech 80]. Le compilateur produit du P-code qui est interprété. Le compilateur n'a pas été modifié; seul l'interpréteur a été légèrement modifié afin de permettre

- le partage de la mémoire et de l'unité centrale entre plusieurs acteurs et le noyau (qui sont tous des programmes Pascal indépendants) dans la même machine - l'interpréteur d'origine ne sait exécuter qu'un seul programme -,
- la communication entre les acteurs et le noyau.

Cette implantation a été réalisée comme un banc d'essai pour l'architecture CHORUS. Elle a prouvé la validité des concepts CHORUS et a montré que la programmation des acteurs était simple. Cette implantation est maintenant utilisée pour des recherches avancées (résistance aux pannes, désignation, protection, etc...).

6.2/ Implantation sur SM90

Une SM90 est une machine multi-processeurs constituée de

- un ensemble de micro-processeurs Motorola 68000 appelés Modules de Traitement (MT) connectés à la fois à un bus global (commun à tous les MT) et à un bus local (propre à chaque MT) à travers une Unité de Gestion Mémoire (UGM),
- un ensemble de mémoires connectées soit au bus global (Mémoire commune - MC),

soit à la fois au bus global et au bus local (Mémoire Local Externe - MLE), soit uniquement au bus local (Mémoire locale - ML),

- un ensemble de micro-processeurs Intel 8080 ou Intel 8086 appelés Modules d'Echange (ME) connectés à une MLE par un bus local. Les ME sont responsables des E/S sur les différents périphériques (consoles, disques, réseau local, etc...). Un ME n'a accès qu'à sa propre MLE.

Une MC est accessible à tous les MT à travers le bus global; une MLE est accessible à travers le bus local par un MT et à travers le bus global par tous les autres MT; une ML est accessible à travers le bus local par un MT.

La configuration maximale d'une SM90 est de 8 MT, 16 ME et 16 Mega-octets de mémoire (pour toutes les MC, MLE, MT confondues).

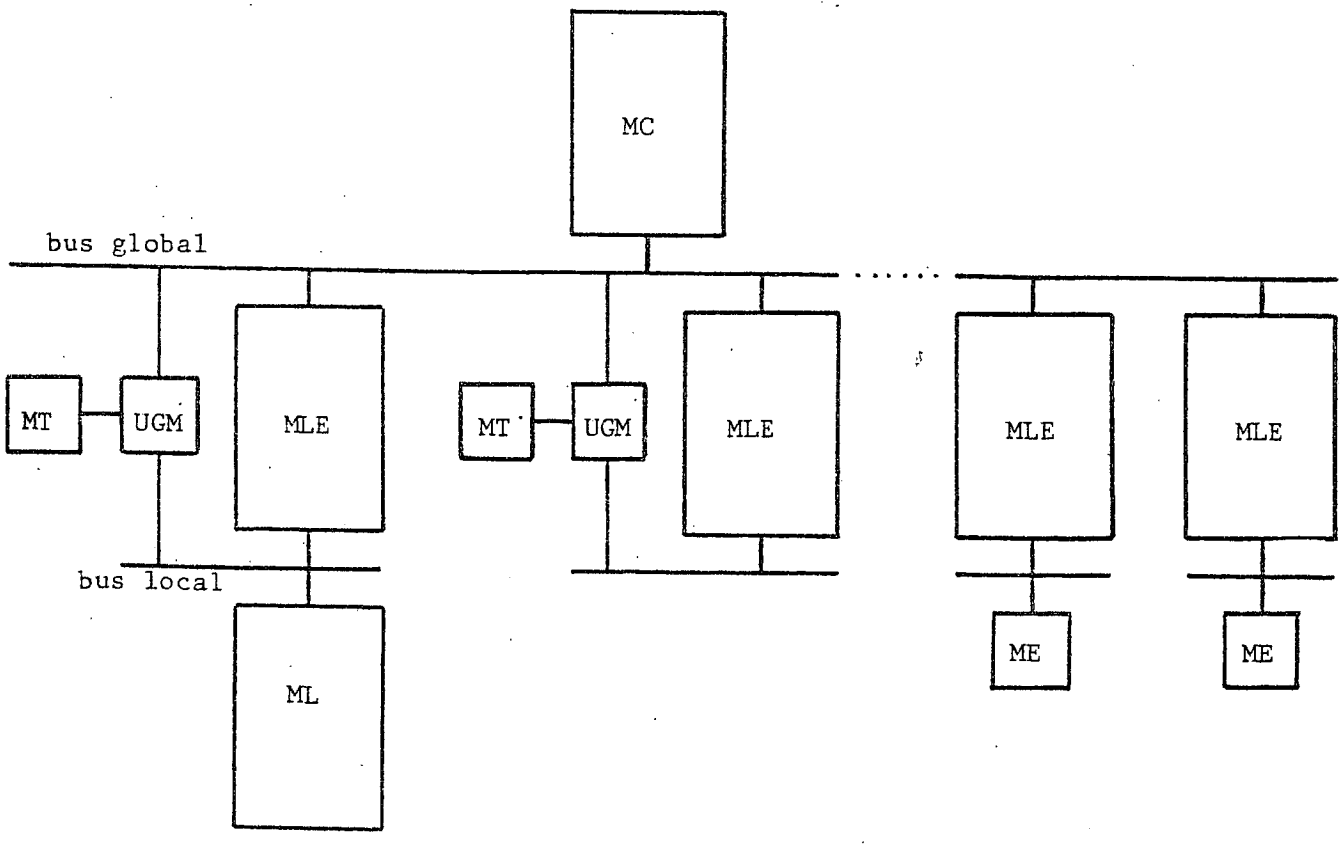

figure 6.1 : MT, ME, mémoires et bus de la SM90

Dans l'implantation de CHORUS sur SM90, un site est un MT et l'ensemble des ML et MLE connectés au même bus local. Les MLE et MC sont utilisées comme un support de communication rapide entre les sites d'une même SM90.

L'Unité de Gestion Mémoire (UGM) est un mécanisme essentiel de la SM90: elle traduit les adresses produites par un MT (considérées comme des adresses logiques) en adresses physiques.

Il y a une UGM par MT. Chaque UGM gère un registre Base-Longueur (BL) et un ensemble de 1024 registres qui décrivent des segment de mémoire physique: chaque registre décrit un segment de mémoire en ML, MLE ou MC (adresse physique du segment, longueur du segment, protection). Chaque processus utilise une "fenêtre" de registres contigus définie par le premier registre (Base) et la taille de la fenêtre (Longueur); la base et la longueur du processus courant sont conservés dans le registre BL; une adresse logique (c.a.d. produite par un MT) est interprétée par l'UGM comme une adresse relative de segment dans la fenêtre courante et un déplacement dans ce segment.

Le temps de traduction d'une adresse est de 50ns.

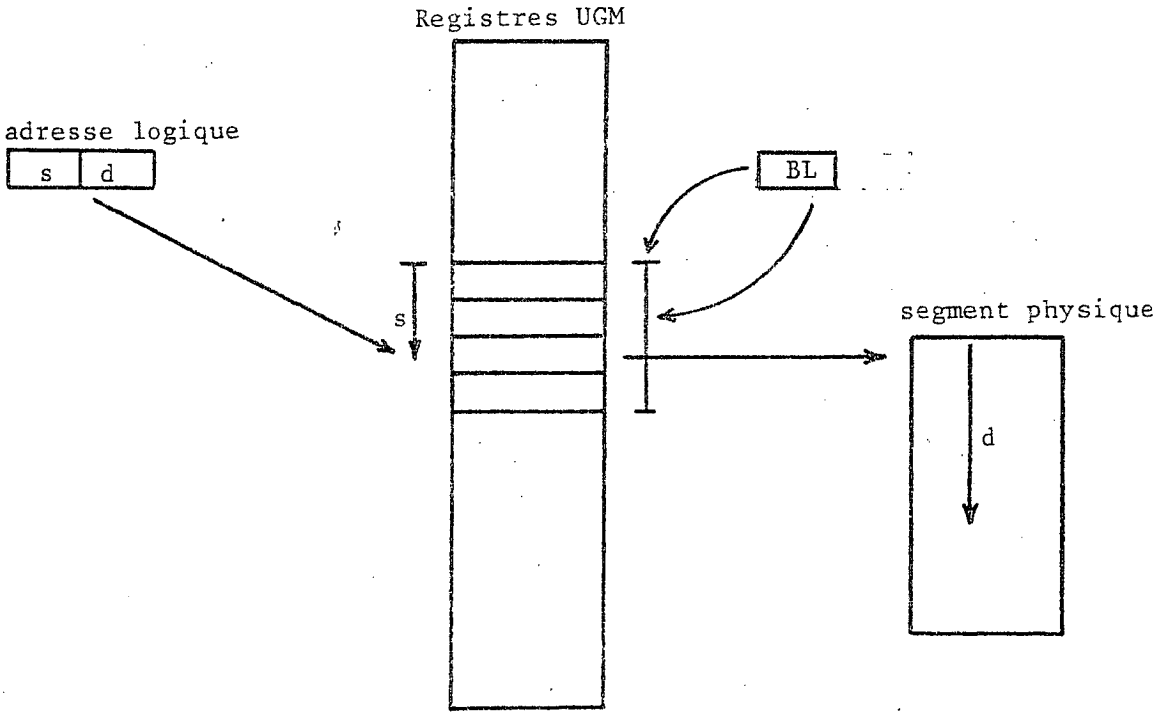

figure 6.2 : Traduction d'une adresse

Cette UGM permet un accroissement des performances:

- un changement de contexte est presque réduit à un changement du registre BL de

la fenêtre de l'acteur courant à la fenêtre du prochain acteur;

- un message est un segment; le transport local d'un message (sur un site) consiste à transporter le registre qui décrit ce message de la fenêtre de l'acteur émetteur à la fenêtre de l'acteur récepteur; il n'y a aucune recopie du message lui-même. Si le message se trouve en MLE ou en MC, il peut également être transporté sans recopie entre deux acteurs situés sur deux sites de la même machine;
- le partage de code entre acteurs est réalisé en installant deux registres identiques dans les fenêtres des deux acteurs;
- le grand nombre de registres de l'UGM (1024 et bientôt 2048) évite d'avoir à faire du swapping de registres.

L'identification d'un message et d'un segment de mémoire permet de transporter du code, des données, des fichiers... entre acteurs exactement comme n'importe quel message, ce qui est très utile pour les acteurs système.

L'implantation de CHORUS sur SM90 s'attache aux performances de CHORUS et servira à valider la capacité de la SM90 à réaliser des systèmes répartis.

6.3/ De l'usage de Pascal

Notre expérience avec le langage Pascal [Guillemont 82b] nous a conduit à deux conclusions :

1/ Il est possible d'écrire des acteurs et presque tout le noyau en Pascal.

2/ De toute évidence, Pascal n'est pas le langage le plus approprié pour CHORUS (ni pour le système ni pour les applications).

Les deux implantations ont utilisé le compilateur de la machine sans aucune modification, mais elles ont utilisé des spécificités de chaque compilateur. Par exemple

- dans le Pascal UCSD (sur Intel 8086), il est possible de définir des segments dans un programme et les liens entre ces segments se font dynamiquement, à

l'exécution des programmes; ceci est utilisé pour réaliser la communication entre les acteurs et le noyau (le noyau est considéré comme un segment commun à tous les acteurs).

- dans le Pascal de la SM90, il est possible de faire de la compilation séparée; ceci est utilisé pour donner aux acteurs une structure d'exécution particulière qui correspond à l'utilisation que CHORUS fait de l'UGM.

D'un autre côté, il manque beaucoup de possibilités à Pascal pour être le langage CHORUS. Par exemple

- Pascal impose que la séquence d'exécution des programmes soit définie par le programme lui-même (avec les SI, TANT QUE, POUR, ...). Dans CHORUS, au contraire, l'enchainement des étapes de traitement est déterminé par les messages reçus avec les conditions de sélection et d'aiguillage: l'ensemble des étapes de traitement d'un acteur ne peut donc pas être entièrement programmé en Pascal.
- Chorus propose une construction d'un acteur en deux étapes (cf paragraphe 2.6) dont la seconde consiste à assembler plusieurs modules en un modèle d'acteur; ceci est une édition de liens entre plusieurs programmes, ce qui n'est pas conforme au Pascal standard.
- La protection doit reposer sur des déclarations communes de types (de messages, par exemple) entre différents acteurs. Pascal n'offre pas cela.
- La communication en Pascal repose sur des appels de procédure. La communication selon CHORUS est asynchrone entre l'acteur émetteur et l'acteur récepteur.

7/ Conclusion

L'architecture CHORUS est bâtie sur un petit nombre de concepts puissants. Dans les différents aspects de cette architecture, nous nous sommes toujours efforcés de choisir des mécanismes élémentaires et simples qui permettent de construire facilement par-dessus toute une variété de stratégies particulières. Ainsi, nous avons obtenu un noyau réduit et un petit ensemble d'acteurs système.

Les implantations et l'expérience acquise nous ont convaincu que CHORUS est une excellente base pour

- apprendre et se familiariser avec la répartition,
- de nouvelles recherches et expérimentations en matière de système réparti.

L'architecture CHORUS a été présentée à un large public d'industriels, d'universitaires, de chercheurs: tout le monde s'accorde sur la clarté et la simplicité des concepts qui rendent CHORUS très attrayant. Ceux qui veulent définir une application répartie en termes d'acteurs apprennent CHORUS en une demi-journée!

En ce qui concerne les recherches et les expérimentations, plusieurs travaux sont en cours, notamment

- la désignation: une implantation des diverses stratégies de désignation (noms globaux uniques, noms fonctionnels, noms de groupe) est réalisée et utilisée [Senay 83].
- la résistance aux pannes: la première expérience d'acteurs couplés [Fabre 82] se poursuit et d'autres stratégies sont examinées.
- des applications réelles sont construites au-dessus de CHORUS; elles nous permettront d'approfondir notre expérience de l'usage de CHORUS.
- le langage et la production de logiciel pour des applications réparties sont des axes de recherche avec une mention spéciale pour l'expression d'activités réparties.

8/ Remerciements

C'est un plaisir de remercier tous ceux qui nous ont aider pour ce papier:

Auteurs, Amis, [Merci]

ANNEXE

Interface de programmation CHORUS

Cette annexe présente un ensemble de procédures qui constituent l'interface de programmation de CHORUS dans l'implantation actuelle en Pascal. Comme cela a été signalé au paragraphe 4.6, ces procédures sont divisées en deux sous-ensembles:

- les procédures "asynchrones" qui construisent des messages et les placent dans la liste des messages qui est transmise au noyau au RETOURNER,
- les procédures "synchrones" qui demandent l'exécution d'un service système de façon synchrone en utilisant le mécanisme d'APPEL_EXTERNE (cf paragraphe 4.1).

Cette annexe donne les procédures d'interface qui concernent le fonctionnement des acteurs, la manipulation des acteurs, des portes et des interruptions; les procédures qui concernent l'accès aux périphériques sont similaires à celles que l'on trouve sous UNIX: elles ne sont pas détaillées ici.

Dans CHORUS, un acteur peut demander simultanément le même service système pour plusieurs entités: par exemple, un acteur peut envoyer simultanément plusieurs messages pour la création de plusieurs portes. Ces messages sont traités dans un ordre non prévisible et par conséquent, les réponses reviendront dans un ordre non prévisible également.

D'un autre côté, un acteur système peut recevoir plusieurs messages de requête pour des services différents sur la même porte: par exemple, l'acteur Gestionnaire de Fichiers reçoit sur la même porte les demandes de lecture et d'écriture sur un fichier.

Ces deux considérations nous ont conduit à adopter la structure suivante pour le texte de tous les messages de demande et de réponse des services système:

Code Service

Code Utilisateur

Paramètres

- le "code service" précise quel service est demandé ou rendu.
- le "code utilisateur" est rempli par l'acteur demandeur; l'acteur système ne l'utilise pas, mais il le copie dans le message de réponse. Ainsi, l'acteur demandeur peut facilement associer la réponse et la demande correspondante: elles ont toutes les deux le même "code utilisateur".
- les paramètres sont fonction du service.

Les procédures d'interface "asynchrones" contiennent un paramètre "Porte locale" que les procédures d'interface "synchrones" n'ont pas: cette "Porte locale" est la porte de l'acteur à travers laquelle est envoyé le message de demande et sur laquelle sera reçu le message de réponse (cf paragraphe 2.1.4); les procédures d'interface "synchrones" utilisent toujours la porte ombilicale de l'acteur pour envoyer et recevoir les messages.

Les procédures d'interface "asynchrones" contiennent aussi un "code utilisateur" que les procédures d'interface "synchrones" n'ont pas: en effet, ces procédures d'interface "synchrones" n'envoient qu'un seul message de demande et l'acteur attend le message de réponse: il n'est donc pas nécessaire de requérir à un "code utilisateur" pour associer ces deux messages.

Contrôle de l'exécution d'un acteur (cf paragraphes 2.2, 2.3.2 et 4.6)

PREPARER (Porte émettrice, Porte réceptrice, Texte du message)

Note: cette procédure empile le message dans la liste.

SELECTIONNER (Porte émettrice, Porte réceptrice)

AIGUILLER (Porte locale, Point d'entrée)

TEMPORISER (Porte locale, Porte émettrice, Porte réceptrice, Délai)

RETOURNER

Note: cette procédure signale la fin d'une étape de traitement; le contrôle est rendu au noyau avec tous les messages à envoyer.

Appel de procédure Externe (cf paragraphe 4.1)

APPEL_EXTERNE (Porte émettrice, Porte Réceptrice, Texte du message, Délai)

Manipulation des acteurs (cf paragraphe 2.4)

CREER_ACTEUR (Porte locale, Code utilisateur, Modèle d'acteur, Site de création)

Note: ceci est la procédure d'interface "asynchrone" de création d'acteur.

CREER_ACTEUR_SYNCHRONE (Modèle d'acteur, Site de création, VAR Nom de l'acteur créé, VAR Nom de la porte ombilicale de l'acteur créé, VAR Diagnostic)

Note: ceci est la procédure d'interface "synchrone" de création d'acteur; elle comporte trois nouveaux paramètres; c'est sur la "porte ombilicale" que l'acteur créateur doit ensuite envoyer le message initial.

DETRUIRE_ACTEUR (Porte locale, Code utilisateur, Nom de l'acteur à détruire)

DETRUIRE_ACTEUR_SYNCHRONE (Nom de l'acteur à détruire, VAR Diagnostic)

AUTO_DESTRUCTION (Porte locale)

Note: cette procédure d'interface est asynchrone! Afin d'être sûr de ne pas être activé pour traiter d'autres messages, l'acteur doit, par exemple, faire Selection (Porte locale, Porte locale).

Manipulation des portes (cf paragraphes 2.3.3 et 2.3.4)

CREER_PORTE (Porte locale, Code utilisateur, Modèle de porte, Nom de la porte à créer, Paramètres d'initialisation)

Note: "Nom de la porte à créer" est un paramètre optionnel: si ce paramètre est présent, la porte créée aura ce nom (pourvu qu'il soit unique); sinon, la porte créée reçoit un nouveau nom unique fourni par le système.

CREER_PORTE_SYNCHRONE (Modèle de porte, VAR Nom de la porte à créer, Paramètres d'initialisation, VAR Diagnostic)

OUVRIR_PORTE (Porte locale, Code utilisateur, Nom de la porte à ouvrir, Priorité de la porte)

OUVRIR_PORTE_SYNCHRONE (Nom de la porte à ouvrir, Priorité de la porte, VAR Diagnostic)

FERMER_PORTE (Porte locale, Code utilisateur, Nom de la porte à fermer)

FERMER_PORTE_SYNCHRONE (Nom de la porte à fermer, VAR Diagnostic)

DETRUIRE_PORTE (Porte locale, Code utilisateur, Nom de la porte à détruire)

DETRUIRE_PORTE_SYNCHRONE (Nom de la porte à détruire, VAR Diagnostic)

Manipulation des interruptions (cf paragraphe 3.2)

ASSOCIATION_INTERRUPTION (Porte locale, Code utilisateur, Interruption)

Note: "Porte locale" est la porte qui doit être associée à "Interruption".

ASSOCIATION_INTERRUPTION_SYNCHRONE (Porte locale, Interruption, VAR Diagnostic)

DISSOCIATION_INTERRUPTION (Porte locale, Code utilisateur, Interruption)

DISSOCIATION_INTERRUPTION_SYNCHRONE (Porte locale, Interruption, VAR Diagnostic)

[Banino 80] J.S. Banino, A. Caristan, M. Guillemont, G. Morisset, H. Zimmermann
CHORUS: an architecture for distributed systems
Rapport INRIA 42, (Novembre 1980), pp. 68

[Banino 82] J.S. Banino, J.C. Fabre
Distributed Coupled Actors: a CHORUS Proposal for Reliability
3rd International Conference on Distributed Computing Systems, Ft Lauderdale, Miami, Florida, U.S.A., (October 1982), pp. 7

[Finger 82] U. Finger, G. Médigue
Architectures multiprocesseurs: l'exemple de la SM90
Minis et Micros no 173, pp. 65, 69

[Grangé 82] J.L. Grangé
A mass transport service on high transmission rate satellite circuits - Some design considerations
(February 1982), pp. 16

[Guillemont 82a] M. Guillemont
The CHORUS distributed operating system: design and implementation
International Symposium on Local Computer Networks, Florence, Italy, (April 1982), pp. 207, 223

[Guillemont 82b] M. Guillemont
Intégration d'un système réparti, CHORUS, dans un langage de haut niveau, Pascal
Thèse de Docteur Ingénieur, Grenoble, (Mars 1982), pp. 250

[Guillemont 84] M. Guillemont
Etude comparative de quelques systèmes répartis
TSI, vol 3, 1 (Janvier 1984), pp. 16

[ISO 82] ISO/TC97/SC16
Open Systems Interconnection - Transport Service Definition
DP 8072, (1982)

[Naffah 80] N. Naffah, B. Scheurer et AL
Description fonctionnelle du réseau expérimental DANUBE
(Juillet 1980), pp. 30

[Senay 83] C. Senay
Un système de désignation et de gestion de portes pour l'architecture répartie CHORUS
Thèse de Docteur Ingénieur, CNAM, (Décembre 83), pp. 200

[Softech 80] SOFTECH microsystems
UCSD PASCAL (TM), version II.0, User's manual
Softech microsystems publication, (February 1980), pp. 400

[Zimmermann 81] H. Zimmermann, J.S. Banino, A. Caristan, M. Guillemont, G. Morisset
Basic concepts for the support of distributed systems: the CHORUS approach
2nd International Conference on Distributed Computing Systems, Versailles, France, (April 1981), pp. 60, 66

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique